Teaching Requirements Engineering to an Unsuspecting Audience

David Callele, Dwight Makaroff
Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada S7N 5C9
callele,makaroff@cs.usask.ca

ABSTRACT

One of a Software Engineer's most important skills is the ability to define the scope of the problem and ascertain the requirements from general and vague specifications. Teaching this skill is known to be difficult and is made more complex because students are conditioned to expect that this portion of programming projects is already complete. This paper reports on experience in teaching a second year computer science class which exposed the need for requirements engineering and gave students an opportunity to engage in the activity. We found that the student response was bimodal, and while some students met the challenge, more felt betrayed by the experience. We conclude that students gained the requisite knowledge using this approach but that a less traumatic approach may produce better results.

Categories and Subject Descriptors

K.3.2 [**Computing Milieux**]: Computers and Education—*Computer and Information Science Education*

General Terms

Management, Verification

Keywords

Software engineering, requirements engineering, pedagogy

1. INTRODUCTION

The present work describes our experiences with introducing Requirements Engineering (RE) principles and techniques to students who were unaware that developing familiarity with the fundamentals of RE was a significant (yet unstated) learning objective for their course. CMPT 214 *Programming Principles and Practices* is a new course in our curriculum, presented in the first term of second year. The course is a requirement for all majors in Computer Science, has an enrollment of approximately 100 students, and is described in the calendar as follows.

> The purpose of this course is to broaden students' view of software development. Topics include introductions to imperative programming languages and scripting languages, programming practices, and tools and techniques for program development and maintenance.

Implicit within *programming practices* is an introduction to RE. The course is presented in a combined lecture and mandatory laboratory format (3 hours and 1.5 hours per week respectively, for 13 weeks). Formal tutorials are not provided but significant support is provided by the faculty, lab instructors, and other students via a discussion board.

The primary objective of the course is to familiarize the students with approaches to problem solving, development methodologies, and development tools to enable the choice of "the right tool for the right job". We want students to be able to understand what they are asked to achieve, and then to solve the task by applying the appropriate development techniques and tools. As a side effect, we expect students to learn skills that lessen the syntactic and process-oriented overhead associated with completing their assignments.

In the remainder of this paper, we review the literature then present a summary of our RE learning objectives for this course. We describe our method for introducing RE in the context of the first major assignment for the class and the student reactions to the technique. We describe our techniques for reinforcement throughout the term, including our capstone scenario, and the results of our efforts as shown by student response. We conclude with a summary of our findings and directions for future work.

2. BACKGROUND

Despite its importance, there is little published work on teaching requirements engineering. Curricula seem to be based around perceived needs for practitioners, often influenced by the personal experience of the course designer and/or the choice of textbook for the course.

Macaulay and Mylopolous [2] examined issues in RE education via a roundtable discussion with other academics at RE '95. At that time, RE education was typically a topic for final-year undergraduates or graduate students. Despite the relatively mature student body, they concluded that teaching RE was inherently challenging:

SIGCSE'06, March 1–5, 2006, Houston, Texas, USA.

> Requirements are variously described by practitioners as 'intangible', 'moving targets', 'inherently inconsistent', 'ever-changing' and a host of other adjectives which fill the average university lecturer with horror... In contrast to this, university courses normally have a prescribed syllabus and strive to provide students with a solid foundation of knowledge, which will guide practice and will direct future learning.
>
> The educational dilemma in teaching Requirements Engineering is to provide the student with this solid foundation in the subject matter while at the same time exposing the student to the inherent uncertainties, inconsistencies and idiosyncrasies associated with real requirements problems.

Despite these challenges, we have seen a steady migration of RE education into ever-earlier introduction to the students. Tuya and Garcia-Fanjul [5] describe teaching requirements analysis to 4^{th} and 5^{th} year students in a simulated production environment. Roscà [4] describes a RE core course in their SE program, seemingly at the 3^{rd} year level. Zowghi and Paryani [6] describe a 2^{nd} year course while our course is also offered in 2^{nd} year.

In all reported cases [2, 5, 4, 6, 3], the literature describes courses with requirements engineering as the primary topic. Role-playing is a universally applied pedagogical tool, with students cast in all roles due to resource constraints. However, requirements engineering is not the primary topic of our course. Therefore, only basic concepts are introduced and any role-playing is performed in conjunction with a faculty member or tutorial leader.

There is solid justification for introducing core RE concepts as early as possible (either explicitly, via formal instruction in RE, or implicitly, as we have done here), despite the complexity of the material. If the RE process does not accurately capture the target for a development project, it does not matter how well the rest of the development process is executed – the output is wrong. The earlier that this lesson is learned by the students, and the more often that it is reinforced, the better. In an absolute sense, success in software development is not measured by how well a process was followed but by the suitability and quality of the result.

3. OBJECTIVES AND ASSESSMENT

The introduction of this course was motivated, in part, by the authors' observations that the effort exerted by upper-year students when completing their computer science assignments was significantly greater than that expected when the assignments were set. Further investigation showed students were weaker than expected at identifying what they were supposed to actually *deliver* for an assignment. As a result, students expended considerable effort on exploratory design and implementation in an effort to identify the requirements. These exploratory efforts increased the time spent on the assignment by approximately 100%.

After considerable review and discussion, the RE-related learning objectives for this course were defined as follows:

1. Recognizing the Semantic Gap: Never assume that you know what something means; take a critical approach to all documentation.
2. Critical Thinking: You can no longer accept that the assignments, as presented, are perfectly stated and without ambiguities or inconsistencies. You must begin to take responsibility for ensuring that you understand what you are expected to achieve.
3. Requirements Elicitation: You are expected to ask questions about the assignment statement.
4. Requirements Validation: The validity of everything in the assignment statement must be confirmed and the assignment statement updated as necessary.
5. Recognizing Emergent Requirements: Try to determine what, if anything, is missing from the assignment statement. In particular, learn to recognize assumptions and implied information.

We follow a relatively strict constructivist pedagogy [1] in the class. Concepts are introduced 'in the small' via examples in the lectures. The concepts are then reinforced in mandatory laboratory sessions before the students are asked to apply them 'in the large' as part of a major assignment. Constant application of principles to scenarios and active discussion makes the classroom a lively place.

Student performance was evaluated via examinations and assignments. Examinations counted for 60% of the final grade, with a lab exam accounting for 13% of the grade. The remaining 27% of the grade was for lab assignments (12%) and major assignments (15%). Twelve mandatory laboratories provided straightforward practice in the use of software tools for productivity and the fundamental components of imperative programming and scripting. Each lab posed a simple task and the lab instructors (a graduate student and a senior undergraduate student) guided the students through a review of the required tools and concepts. These concepts were introduced earlier in the classroom. Student deliverables comprised software artifacts and test results, including a revision history of their work. Major assignments formed a significant evaluation component. Students had about four weeks to complete a substantial analysis, design, implementation, and testing effort. Examinations were open-book and required competence in simple synthesis of course concepts and programming fundamentals.

Given that this is a 2^{nd} year course, we are only able to provide a gentle introduction to these learning objectives. Our techniques are presented later in the paper, but first we present a review of the role of the customer when teaching requirements engineering.

4. THE ROLE OF CUSTOMER

Requirements engineering generally assumes the presence of a customer (or stakeholder(s)). Polajnar and Polajnar [3] state that this customer can be 'real' (a stakeholder with a true interest in the outcome of the process) or 'virtual' (a stakeholder that simulates a true interest in the outcome of the process). While we agree, we prefer to apply a more finely-grained categorization when identifying customers for pedagogical purposes. Table 1 is a summary of the resources we have used in this, and other, software engineering classes. All but one resource simulates the customer experience and we present brief commentary on each entry.

Many useful learning experiences can be constructed without a customer of any form – the first assignment in this

Table 1: Customers

None	Problem statement serves as customer communication.
Student	Undergraduate from within the class. Undergraduate who has recently taken the class, usually one of the top performers. Graduate, ideally with large-scale or industrial experience.
Faculty	Without RE experience. With RE experience.
Faculty	Instructor and Evaluator for the course.
Industrial partner	Support from local industry.
Real customer	Best learning experience, highest risk.

class creates an intense "customer experience" with only an assignment statement. However, a customer of some form generally enhances the entire learning process.

As noted earlier, due to cost and time constraints, customers are most commonly simulated by students. The risks are substantial – the student customer must separate their role as customer, educator, and (fellow) student. While some students, particularly those with industrial experience, are able to simulate a customer in an acceptable manner – most are not, for they lack necessary domain experience.

Faculty are usually a better choice than students, particularly if they can use a research project as the basis for a (relatively) real customer experience. We have an interesting observation: faculty *without* requirements engineering experience often make better customers. We have noted that experienced practitioners tend to impose their perspectives on the process in their zeal to enhance the student learning experience. This can lead to pedagogical conflicts or the customer dominating the student development team.

Industrial partners can be very effective simulated customers. They bring their experience with real customers to the classroom, are unlikely to interfere with the educational goals, and tend to view the process as an early recruiting investment. Unfortunately, it is very difficult to find a partner that can make a time commitment to a classroom schedule.

There are significant pedagogical risks associated with using real customers – they are relatively uncontrollable and their needs may not match the learning objectives for the course. Their schedules, particularly delivery schedules, are often too tight to be used in a classroom setting. However, there is a sense of *risk* engendered within a development team when working on a real project [2] that is difficult to create in simulated scenarios.

We believe that simulated educational experiences should strive to recreate this sense of risk and have applied this principle in our classroom. In the absence of the necessary resources for role-playing [6], carefully crafted scenarios can selectively introduce risk-generating issues while maintaining pedagogical integrity. For example, a set of requirements can be presented to the students in the guise of an assignment statement. The instructor can choose to introduce one or more issues (*e.g.* missing requirements, contradictions in the stated requirements) while keeping all other elements of the scenario under strict control. Given that the instructor is (usually) the assigner of grades, they are able to induce a sense of *uncertainty* and *risk* in the students (just as if they were working with a real customer) by manipulating the number of issues and the grading scheme.

5. WELCOME TO THE REAL WORLD

The first major assignment for the class was to write a bash shell script. The students were presented with a detailed narrative description of the task that the shell script was to perform. They were also presented with a table of command line options that the shell script had to support.

The students had received formal instruction introducing all shell utilities and constructs necessary for successfully completing the assignment. Examples, directly related to the assignment but not explicitly identified as such, were also presented to the students.

To facilitate instruction of the RE principles, we deliberately placed a contradiction in the assignment statement. The body of the task description stated that the shell script was to support standard shell constructs such as I/O redirection. However, the table of command line options explicitly listed a "-I" option that identified an input file name and a "-O" option that identified an output file name. This is a classic case of contradictory customer requirements: It is not reasonable to support I/O redirection and explicit file I/O identification within a single utility.

We reserved 5% of the total marks as a reward for those students that correctly identified the contradiction and approached us for clarification. Unfortunately, none of the students came to us for clarification. Upon inspection of the submissions, it did appear that 3 of 97 students identified the contradiction and attempted to design around it.

Inspecting the contents of the message board revealed that students focused their immediate attentions on those requirements that appeared to be explicitly identified; for example, how to parse the command line. It was only comments posted in the last 24 hours before the assignment was due that showed that the students were pursuing classic RE efforts such as defining terms and specifying the format for various I/O functions. We saw no evidence on the message board that any of the students considered more advanced topics such as looking for implied information or checking for contradictions within the assignment statement.

Immediately after the assignments were submitted, we identified the contradiction to the students in a classroom discussion. Without identifying the marking scheme, we pointed out to the students that none of them had actually answered the question that we asked of them. Instead, they had answered something else – if we were so inclined, we could justify giving them a grade of 0%.

Reaction was swift and vociferous. There were those with a sense of entitlement ("you *have* to give me part marks") and those with outrage ("you *can't* give me an assignment with an error in it, that's not fair!"). Most of the students only heard the "grade of 0%" part of the comment and immediately assumed that we were going to fail every last one of them. This selective hearing process illustrates just how ill-prepared the students are to attempt RE tasks with a real customer and why introducing this material in a learning environment is so important.

The following student comment, submitted via an anonymous class evaluation mechanism at the end of the term, is a reasonable summary of student reaction.

> The assignment specs kept getting changed, so you would work some 20 odd hours, then have to SCRAP everything because we were told not to work on the project a certain way. I under-

> stand that they want us to learn how to ask questions, but this is NOT real life it is UNIVERSITY and an assignment should be set in stone.

Enabling the students to understand that part of their assignment was to determine exactly what that assignment was *really* asking for was a challenge for all concerned. This was the first time that most of the students were ever expected to take responsibility for analyzing their assignments for completeness and as the prior quotation illustrates, this seemed to offend their sense of right and wrong. Some felt that we were negligent, others that we were deliberately trying to set them up for failure – relatively few understood just how important these lessons are.

The discrepancy between perception and reality was marked. As the student's comprehension of the requirements increase, many perceived that the specification was changing despite the fact that it was not. Most of the students that felt the specifications were changing were those who did not participate in classroom or electronic discussions of the assignment.

6. PRACTICAL GUIDANCE

The students were then introduced to some simple syntactic and semantic analysis techniques to use in the requirements identification and elicitation processes. At the beginning, we introduced actors and use-cases (without identifying them as such) by grammatical analysis of the documentation on a sentence-by-sentence basis. Each sentence was, if necessary, rewritten as a simple declarative statement of the form: *subject-verb-object*. The subject and object nouns were then identified as potential actors and the verb as a potential use-case for the subject. The students were also instructed to be careful to look for synonyms during this analysis to ensure that consistent terminology was used throughout the document.

At a semantic level, the students were presented with examples of implications and assumptions. We introduced the concept of *proactive paranoia*, the need to question the existence of almost everything in the document and provided them with examples of the kinds of questions that *we* would ask the customer, if we were presented with the same documentation set.

Within the laboratory setting, the laboratory leaders assumed the role of the customer for the students and provided them the opportunity to practice their skills at requirements elicitation and identification.

Finally, the students were required to rewrite the assignment statement, including any new information gathered during their requirements elicitation and identification efforts, in as unambiguous a manner as possible. The mean grade on this task was 83% with a standard deviation of 22% – clear evidence that the majority of the students had learned their lesson. Unfortunately, approximately 10% of the class still had significant misunderstandings about the basic concepts and processes. By the end of the term, it was clear that we had been unable to motivate or reach this group of students.

Our students come to us preconditioned by a lifetime of "continuous evaluation" and feedback like "there is no failure, unless you don't try". Concepts such as mandatory submission formats were totally foreign to them. For example, we provided formal instruction on concepts related to meeting contractual requirements, including describing scenarios where the work had to be submitted to the customer in a very particular format or the work would be rejected. At best, they would still have to resubmit in the correct format, or at worst, they would not be paid, despite their efforts. In the words of the great Jedi warrior, Yoda: "Do, or do not. There is no try." [1] – a near complete reversal of their prior conditioning.

To emphasize the necessity to identify and meet *all* requirements, not just functional requirements, we instituted a mandatory submission format for the second assignment. Only 9% of the students complied. Rather than reject the remaining assignments, we allowed the students to resubmit their work in the required format but with a performance penalty (similar to the technique used by Tuya and Garcia-Fanjul [5]). A 20% penalty was applied to all resubmissions, effectively blocking the student from receiving an 'A' on the assignment. When we addressed the issue in review with the students, most reacted negatively. A typical comment, submitted via the anonymous class evaluation mechanism:

> . . . what's the point of spending 40 hours on something that you're going to get 40% on just because you forgot one line, or handed in the directory containing the assignment instead of the files within the directory?

Enabling the students to understand that executable code is only one part of their deliverables was difficult. Continuous evaluation certainly makes it difficult to teach personal responsibility and a requirement that some elements *must* be right before the rest can be evaluated.

7. REINFORCEMENT

We continued to reinforce the message "ask questions" throughout the remainder of the class. At times we deliberately inserted ambiguities or contradictions into our materials as an instant test of the student uptake of the concept. Our perception was that a distinguishable subset of the student population noticeably improved their ability to detect these issues.

In general, the students felt that the first assignment taught a harsh lesson. Their emotional response was strongly bimodal. One group felt intellectually challenged and eagerly responded to this stimulus. The second group felt abused, and put upon, and embraced defeat. We found it quite difficult to reach the group of students that gave up and get them motivated again.

We followed through on building the clarification skills with the second assignment in the course. Assignment 2 was a C function library for manipulating doubly-linked lists. The level of algorithmic difficulty was similar to assignments from 1^{st} year, but the implementation was in C rather than Java. The technical differences were the use of pointers rather than references, providing a statically allocated block of memory for data structures, and using the development environment to create a library.

We invited students to ask questions to clarify their understanding of the requirements within the classroom environment. A student who asked a question was required to type up the question and our response then forward the material to us. After double-checking their submission for

[1] Star Wars Episode V: The Empire Strikes Back

accuracy, we then added the information to the assignment statement in a revision history format.

With the design/development methods available and suggested programming tools actually being used, the better students added complexity to the assignment – rather than reducing the time required, time was kept constant and extra work was performed. When asked, one of the students said that they did not believe that the assignment could actually be that easy. In contrast, a different student stated (in the anonymous end-of-term course evaluations) that this was "...an assignment that was so unclear (and yet advanced) that many students still don't understand it."

8. A FINAL LESSON

An in-class exercise was scheduled for the second last lecture of the term. The purpose of the exercise was to allow the students to practice their newly acquired skills with a simulated customer. One author played the role of the customer, the other author the role of the team leader, and the students composed the development team.

The exercise began with the customer stating their needs: "I need a program to manage tournaments." Then, the customer walked out the door, much to the astonishment of the students. The team leader then stimulated conversation within the classroom and captured the students' questions on the board. The questions were typical of requirements elicitation: "How many competitors must we support?", "What determines winning and losing?" and "How do we display or disseminate the results?"

After 15 minutes, the customer returned and stated that he was ready to answer questions. He was immediately bombarded with questions about sports and rules, judging and prizes, etc. After pausing for a few moments, he looked at the class and stated (in an outrageously bad accent) "What is all this stuff about sports? I work for the Lucky Mint Manufacturing company and we make mints. In order to keep our inventory fresh, we need to turna-de-mints on a regular basis. I need a program to manage the mint-turning robots, not something for games."

After the groans, catcalls, cheers and jeers subsided we felt truly successful – the students would not soon forget that, when performing requirements engineering, you must question everything!

9. POSITIVE FEEDBACK

The experiences reported in the last few pages may make it seem like the course was not a success. This was not the case – approximately 25% of the student body reacted very positively to the course content and delivery mechanism. The intellectual and emotional growth we witnessed in some of the students was simply amazing. Here are just a couple of their comments:

> This course was incredible, who ever thought of it is a freaking genius. Seriously, this course is everything I wanted to learn ...
>
> Give a raise to whoever thought of having this course, we need more courses like these, really we do!

While it is still too early to quantify the long-term benefit of this technique, we have observed that the students are still talking about their experience in this class nearly a year later – a sure sign that our approach is, at the very least, memorable. We believe that the students are now more aware of RE principles and some of the fundamental RE techniques which should allow them to recognize the need to apply them in the future. A colleague, teaching a 3^{rd} year software engineering course, commented that the students' approaches to the materials in his class were noticeably more mature after the introduction of this course. The students' more proactive approach in clarifying assignment specifications has been noted by the instructor in the programming languages course.

10. CONCLUSIONS AND FUTURE WORK

Requiring students to perform critical analysis of their assignment statements is a major, if not traumatic, shift in their thinking. Requiringg 2^{nd} year students to assume responsibility for determining what they have to submit divided the class in two – a group of students who rose to the challenge and a group of students that gave up. The emotional maturity in our student population may not have been high enough for this pedagogical approach. This technique can create an emotionally-charged atmosphere in the student body and a degree of finesse in classroom management may be required.

Many students found it difficult to adjust to evaluation methods such as mandatory submission formats as a requirement that must be met before marking can begin *and* receiving marks proportional only to the quality of the result after a lifetime of receiving "marks for effort". We have proposed changes to the 1^{st} year curriculum to introduce these submission requirements in a less dramatic manner.

We continue to investigate alternative pedagogies in an effort to identify a mechanism that replicates (some of) the stress associated with requirements engineering, yet less stress than the current technique. We are also continuing to explore the role of *evaluator* as customer in a controlled learning environment.

11. REFERENCES

[1] Peter E. Doolittle. Constructivism and Online Education. In *1999 Online Conference on Teaching Online in Higher Education*, pages 1–13, 1999.

[2] L. Macaulay and J. Mylopoulos. Requirements Engineering: An Educational Dilemma. *Automated Software Engineering*, 2(4):343–351, September 1995.

[3] D. Polajnar and J. Polajnar. Teaching Software Engineering through Real Projects. In *WCCCE 2004 Western Canadian Conference on Computer Education*, pages 83–90, Kelowna, B.C., Canada, May 2004.

[4] D. Rosca. An Active/Collaborative Approach in Teaching Requirements Engineering. In *30th Annual Frontiers in Education Conference*, pages 9–12, Kansas City, MO, October 2000.

[5] J. Tuya and J. Garcia-Fanjul. Teaching Requirements Analysis by Means of Student Collaboration. In *29th Annual Frontiers in Education Conference*, pages 11–15, San Juan, Puerto Rico, November 1999.

[6] D. Zowghi and S. Paryani. Teaching Requirements Engineering through Role Playing: Lessons Learnt. In *11th IEEE International Requirements Engineering Conference (RE'03)*, pages 233–241, Monterey Bay, CA, USA, September 2003.