Teaching Requirements through Interdisciplinary Projects

Deepti Suri, Eric Durant

Department of Electrical Engineering and Computer Science
Milwaukee School of Engineering
1025 North Broadway
Milwaukee, WI 53202-3109
{suri, durant}@msoe.edu

Abstract

Requirements Engineering (RE) is the process of determining, analyzing, documenting, validating, and maintaining the services and constraints of the systems that need to be designed. Because of the high importance of RE in the design of software systems, the need to expose students to RE in the undergraduate Software Engineering and CS curricula is getting more attention. Working in unfamiliar domains, being cognizant of ethical issues, and having to deal with ambiguous and conflicting customer requirements are some of the challenges that students face in a course like this.

The authors have added a practical element to a third year undergraduate course in requirements for software engineering (SE) majors through a quarter-long project in which the students work with clients who have product domain knowledge but often no formal experience in RE. The clients are biomedical engineering (BE) student design teams. This allows interdisciplinary collaboration, exposes the SE students to eliciting requirements in an unfamiliar domain, and exposes the BE students to a formal requirements process. The authors discuss what they learned from this collaboration, which is currently in its second year.

1. Introduction

The chances of a product being developed on time and within budget are dependant on thorough and precise analysis of the client's current situation and needs. Informally, the client's needs are also called "requirements". A "requirement" is a specification of what should be implemented by a product. The IEEE standard defines a requirement as "a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component" [2] [3].

Requirements are primarily of two types: functional and non-functional. Functional requirements relate to the actions that the product must carry out in order to satisfy the fundamental reasons of its existence. Non-functional requirements are the desirable properties/qualities that the product must have [4] [7]. These are the characteristics that make the product fast, usable, portable, reliable, attractive, etc.

Requirements Engineering (RE) is the process of determining, analyzing, documenting, validating and maintaining the services and constraints (i.e. the functional and non-functional requirements) of the systems that need to be designed for the client. The use of the term "engineering" implies that systematic and repeatable techniques are used to ensure that the requirements are complete, consistent and relevant. One early intermediate product of RE is the

Software Requirements Specification (SRS) document [1] that describes all the externally observable behaviors and characteristics expected of a software system. A quality SRS is one that contributes to successful and cost-effective creation of software that solves real-user needs and usually incorporates the viewpoints of all the stakeholders who have an interest in the product.

The cost of discovering a requirement during construction of the product, or, worse when the client starts using the product, is expensive and inefficient and yet a large number of companies typically spend only about 10% of the total time allocated for the project on requirements gathering, 10% on specifications, 15% on Design, 20% on Coding and 45% on Testing (and hence it is said that a typical project approximately follows the 40-20-40 rule). By the time the project is typically done (usually over budget and late), many companies discover that 50-80% of their total budget is spent on rework. It has been discovered that successful projects follow the 60-15-20 rule where 60% of the total time is devoted RE & Design [9].

Because of the high importance of RE in the design of software systems, RE is a required junior level course in the undergraduate Software Engineering (SE) curriculum at the Milwaukee School of Engineering (MSOE). Such a course (SE-3821[1]) was developed and taught for the first time in our SE curriculum during the Winter Quarter 2000-2001 [8]. The students are first introduced to the importance of requirements in SE-283 (Introduction to Software Verification), and SE-280 (Software Engineering Process). The concepts of RE learnt in SE-3821 are reinforced in SE-380 (Principles of Software Architecture) and the process is scaled up for the students in their three-quarter experience of "Software Development Laboratory"[5], where the students work on large-scale projects in a "real-world" setting.

2. Curricular context

The academic schedule at MSOE is based on a quarter system with three quarters in an academic year. Each quarter involves ten weeks of instruction with the eleventh week devoted to final exams. Typical software engineering courses are three or four credits, and most have an associated laboratory session.

The undergraduate software engineering program at MSOE [6] began operation in 1999 and had its first graduating class in spring 2002. The SE program was visited by the Accreditation Board for Engineering and Technology (ABET) in September 2002 and is one of the first accredited SE programs in the United States.

The software development laboratory course sequence begins in the winter quarter of the junior year and extends through the fall quarter of the senior year. Upon entry into the software development laboratory (SDL), students have already completed courses in programming, data structures, algorithms, design patterns, embedded systems software, requirements, testing and individual software process. During their time in the lab, students simultaneously take courses in software architecture and formal methods.

After completing the software development lab sequence, students also work in teams on a two-quarter capstone senior design project, which may be multidisciplinary in nature.

[1] The course numbers are based on version 2.0 of the curriculum.

3. Background Information on the Course

As mentioned earlier, a course on Software Requirements and Specification was first developed and during the 2000-2001 winter quarter [8]. Since then the course has continued to evolve. Since the first offering people working on real-world projects outside of the SE curriculum context have acted as stakeholders to SE student groups for their requirements projects, culminating in an SRS. Although these external collaborations were largely positive experiences, they had some drawbacks, including great difficulty scheduling and traveling to meet with the external stakeholders. In the 2003-2004 offering, other students were used as stakeholders for the first time. Specifically, biomedical engineering (BE) students in the third year of a four year research, development, and design project acted as domain experts with SE students acting as requirements experts.

Having SE juniors work with other juniors is desirable since it increases their comfort level. However, the drawback is that even though the BE project course sequence has an extended market and technology research phase, many aspects of the design have already been drafted by the time the SE students begin working with the BE students. Although not ideal, the timing of the collaboration is the best feasible option. The second year BE students are not far enough along in their research to provide the needed input on requirements and constraints. In fact, the timing is appropriate for documenting requirements that are unambiguous and measurable, since the BE students have developed the appropriate domain knowledge by the third year.

This collaboration was planned before the quarter began by the two faculty teaching the requirements course (the authors) and the faculty member advising the third year BE students. The BE faculty member and one of the authors had worked together on a requirements projects earlier, though in a different context. Both of them believed that such an experience would be beneficial to all the students and that is how the collaboration got started. The SE students were to benefit by practicing the requirements process in an unfamiliar domain. In fact, the projects involved products that had only very small software components, but instead consisted primarily of processes and hardware. The BE students were to benefit through exposure to a formal requirements process that greatly increased the chances of ending up with a product that meets the goals of the work while minimizing wasted effort.

4. Methods

The main philosophy of the course is “learning by doing”. The goal is to provide some exposure to student teams in eliciting and specifying requirements. We also want to expose the students to a domain with which they are not familiar, so that they do not have any preconceived notions about the project. The hardest part of eliciting requirements is to be able to understand the terminology and the language of the stakeholders. We believe that students do not appreciate the difficulty of eliciting requirements if they are placed in a domain with which they are already familiar.

The BE student teams (also the primary stakeholders) were giving a project presentation to the BE instructors in the second week of the quarter. The SE students were encouraged to listen to the presentations and submit their preferences regarding the project they wanted to work on. The SE student teams were formed primarily based on the student preferences by the end of the week.

The BE students also provided all of their project documentation to the SE student teams. The SE student teams were expected to sort through the information and then schedule an elicitation meeting with the stakeholders to gather additional information about the project. Since there were

two SE teams to one BE team and there was no scheduled common time for these teams to meet, the students had to arrange their own meeting times and this at times was a problem. A significant part of the communication took place via e-mails. We, as instructors, encouraged the SE teams to share information among themselves even though they had to work independently. This was done to prevent the BE teams from answering the same questions over and over again.

During the course of the quarter, the students worked on four assignments (detailed later in this section) based on the Volere process described in [4], which were sent to the BE student for review as soon as they were submitted. The assignments were reviewed by the instructors and feedback was provided to the students. The requirements were graded on the criteria of completeness, lack of ambiguity, testability, and readability. A lot of emphasis was placed on the fact that all assumptions and constraints were listed, the glossary was up-to-date, and all alternate and exception flows were considered when enumerating a use case.

The four assignments were:

Assignment 1	Project Blastoff (i.e. enumerate goals, identify stakeholders, enumerate constraints & risks, contextual diagram)
Assignment 2	Specify the major use cases for the system.
Assignment 3	SRS for the project.
Assignment 4	Complete and finalize the SRS. Final project presentations.

More details regarding the details of these assignments can be found at http://people.msoe.edu/~durant/courses/se382/outline.shtml

5. Results

This course was first offered in the winter quarter of 2000-2001 and has been offered every year since at MSOE, having transitioned to the fall quarter in 2004. It has evolved over time based on assessment and feedback received from both students and stakeholders. There is a pattern of students not viewing this course favorably at the end of the quarter. We believe that it is because this course is very different from all the other courses that the SE students are exposed to in the SE curriculum.

- Students do not do any coding in this course. Instead they spend a lot of their time writing documents and specifying the project requirements completely and correctly. This was their first experience in doing so with any kind of rigor.
- The students worked as teams on the same project throughout the quarter. The projects were solicited from a domain that they are unfamiliar with. Apart from going through the daily rigors of the course; the students were also dealing with external stakeholders, learning about a new domain as well as learning how to specify “good” requirements.

There is a huge turnaround in attitudes and students seem to understand the role and importance of specifying requirements correctly either during or after their experience in SDL. The SDL is a three-quarter course sequence in the junior and senior years, designed to provide the students a “real world” experience in an academic setting. This arrangement provides students an opportunity to work in teams on ongoing large-scale projects [5]. Since the students are working on a project for which they may not have gathered the requirements themselves, they understand the importance of unambiguous specification. The students in the SDL also have a budget of only

ten hours per week. At some point each student also experiences the havoc that changing requirements midway through development can cause.

Student feedback at the end of the winter 2003-2004 quarter provides similar results. Some student comments are included.

Things you liked about the course:

- It was interesting to discuss software on a non-code level.
- Progression to a final SRS, material seems relevant.
- The level of detail that goes into requirements and the logic of determining requirements.
- A very important course. Able to pick up some of the concepts and vocabulary.
- Seems like it will be useful.

Things, which could be improved:

- BE project did not work well.
- Integrate the SRS type homework into examples in class.
- Better book. More mature (further along) design teams to work with. Team teaching – not so good.
- BE students!
- Get rid of projects.

At MSOE, each course has a set of objectives that are published and distributed to the students at the beginning of the term. At the end of the term, students are asked to evaluate themselves on how successful they were at meeting each objective and then evaluate the course on helping them meet those objectives. The scales used are given below.

Rating	**Students success in meeting objective**	**Course's assistance in objective**
1	Did not meet this objective at all	Not at all helpful
2	Met this objective only in part	Not very helpful
3	Minimally met this objective	Adequately helpful
4	Met this objective completely	Very helpful
5	More than met this objective	Extremely helpful

The scales are not correctly balanced, but remain in use so that many years of historical data can be directly compared. The negative bias on rating 3 is stronger for the student success scale, so means of at least about 3.5 are desired on the student success scale, while means of at least about 3.0 are desired on the course objective scale.

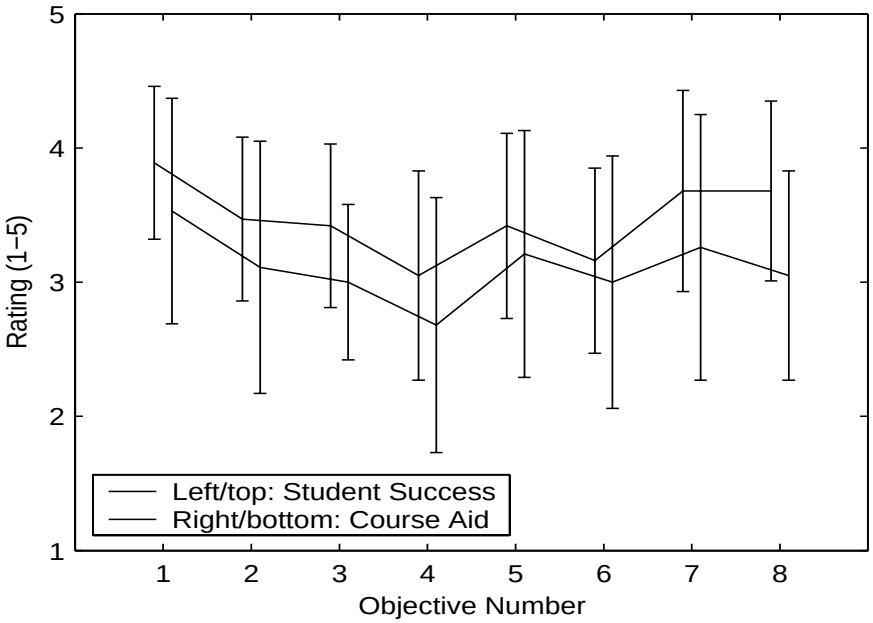
The course objectives SE-3821 are presented in Table 1. The average scores of all the students along with the standard deviations for the Winter 2003-2004 quarter for all eight objectives are presented in Figure 1. Out of 20 students, 19 completed the self-assessment of objectives.

Table 1: Course Objectives for SE-3821

O1	understand the role of requirements engineering in a variety of software development models
O2	be able to elicit requirements from system stakeholders and to overcome common obstacles to the elicitation process
O3	be able to analyze and negotiate software requirements
O4	be able to specify software requirements with use cases, formal methods, and other documentation techniques

O5	be able to specify requirements that are verifiable, traceable, measurable and testable
O6	be able to verify that specified requirements are accurate, unambiguous, complete and consistent
O7	understand the importance and common methods of managing software requirements
O8	be able to communicate software requirements in written documents and oral presentations

Figure 1: Course Assessment Information for 2003-2004 winter quarter



6. Discussion

The student comments presented above highlight several, mostly positive, themes. First, the appreciation of students for the non-code, early development activities of software seems to be increasing; hopefully this is emphasized by conducting a project in a domain not tightly coupled to software. There were some negative comments about collaborating with the BE teams; although eliminating the off-site collaborations eased some aspects of collaboration, students dealt with varying degrees of success with coordinating a large number of schedules just as one must in the real business world. The comment that the BE teams should be further along is difficult to interpret, but suggests that the particular student does not understand or is not comfortable with the place of requirements early in the engineering process, despite many discussions about this during lecture. The comment on team teaching is similarly confusing; the professors attended nearly all of each other's lectures and copied one another on emails to students to ensure that a unified, consistent message was being delivered to students despite approaching the subject with two very different backgrounds.

Student self-assessment on objectives 4 (3.05) and 6 (3.16) are noticeably lower than the target of 3.5. Objective 4 can probably be explained by the course not explicitly using material from the formal methods class that students took during the previous quarter. Objective 6 generates more

concern. It covers four aspects of good requirements, which are exercised throughout the class. Completeness of requirements presents special challenge to experienced professionals, so it is not necessarily a poor outcome that students are more tentative in their self-evaluation of this area. It is interesting that the smallest bias of self-success over course aid occurred for objective 6 (0.16), while the largest occurred for objective 8 (0.63). This may reflect the students' confidence in presenting technical material from several previous courses.

7. Summary and Future Directions

In summary, requirements in general, and the authors' project approach in particular, emphasize different skills than those with which most engineers have the greatest comfort. The emphasis on understanding a new domain and finding requirements before doing design enables some previously apparently mediocre students to excel and causes others to stumble.

Some of the most consistently raised areas of concern were with the nature of the BE student side of the collaboration. To that end, we are working with the advisor to the current third year BE students to more tightly integrate them. This will involve a presentation by the SE faculty to the BE students early in the quarter to communicate the goals and benefits of the collaboration. Also, the SE students will provide the BE students, as stakeholders, with copies of their four reports throughout the quarter, allowing the BE students to give feedback and use the evolving SRS to help with their design.

8. References

[1] Davis, A., Overmayer, S., et al.,"*Identifying and Measuring Quality in a Software Requirements Specification"*, Software Requirements Engineering, Second Edition, IEEE Computer Society, pp. 194–205.

[2] Jackson, M., "The Meaning of Requirements", *Annals of Software Engineering*, Baltzer Science Publishers, Vol. 3, pp. 5–21, 1997.

[3] Nuseibeh, B.A., Easterbrook, S.M. and Finkelstein, A.C., "*Requirements Engineering: A Roadmap*", The Future of Software Engineering (Companion volume to the proceedings of the 22nd International Conference on Software Engineering (ICSE'00)), IEEE Computer Society Press.

[4] Robertson, Suzanne and Robertson, James, *Mastering the Requirements Process*, Addison-Wesley, 1999.

[5] Sebern, M.J., "The Software Development Laboratory: Incorporating Industrial Practice in an Academic Environment", *Proceedings of the 15th Conference on Software Engineering Education and Training*, Covington, KY, USA, February 25-27, 2002, pp. 118-127.

[6] Sebern, M. J. and Lutz, M. J., "Developing Undergraduate Software Engineering Programs," *Proceedings of the 13th Conference on Software Engineering Education & Training*, Austin, TX, USA, March 2000, pp. 305-306.

[7] Sommerville, I. and Sawyer, P., Requirements Engineering: A Good Practice Guide, John Wiley & Sons, 1997.

[8] Suri, Deepti, "Introducing Requirements Engineering in an Undergraduate Engineering Curriculum: Lessons Learnt," *ASEE Annual Conference and Exposition Proceedings*, CD-ROM, Paper No. 560, Montreal, Canada, 2002.

[9] Van Vliet, H., *Software Engineering: Principles and Practice*, Second Edition, John Wiley, 2000.