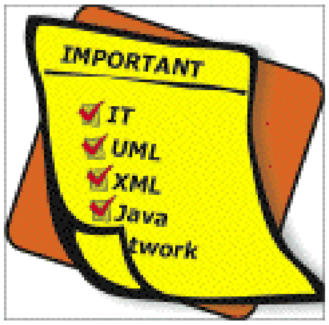
Crossroads
The ACM Student Magazine



Requirements Engineering: Closing the Gap Between Academic Supply and Industry Demand

by ***Kristina Winbladh***

Introduction

Recent trends in the information technology industry call for substantial changes in computer science education. As an effect of the recession, the increase in the number of computer science graduates, and the outsourcing of information technology (IT) work to other countries, the unemployment rate among engineers and especially among computer science professionals is increasing [11]. In this economic situation, it is imperative that computer science students are well prepared before entering the work force; new graduates must understand what skills the IT industry is seeking.

Computer science education in the United States is currently being criticized by the IT industry for not providing graduating students with sufficient management, problem solving, and communication skills [7]. Current research in software engineering and computer science education is targeting this criticism to understand what industry demands, what educational institutions supply, the gaps between supply and demand, and changes that could improve computer science education.

The software development process consists of several steps, and those steps can further be divided into smaller sub tasks. Rather than focusing on the entire development process, this article will consider the initial step of the software engineering process and the requirements elicitation step in more detail. Requirements elicitation is a part of a larger activity called requirements engineering. In order to address the elicitation process, requirements engineering must also be addressed. We focus on elicitation of requirements here because it is an essential part of the development process. The objective of this article is to alert and update students on current research and to accentuate the significance of research in this area. After all, the outcomes of this research could affect the future of many graduating students. The more graduates are aware of industry's viewpoint, the better prepared they can become for their future careers.

What Is Requirements Engineering And Why Is It Important?

Requirements engineering is the activity that involves capturing, structuring, and accurately representing the client's requirements in a manner that can be effectively implemented in a system that will conform to the client's specifications. Good requirements engineering is crucial to the software industry, as the costs of fixing errors due to fallacious requirements increase exponentially the longer they remain unnoticed [9]. Reports have shown that software projects tend to waste tremendous resources on rewriting code that has been implemented due to mistaken assumptions about the client's requests. Another central concept in requirements engineering is the notion of requirements being dynamic, which means that they will continue to evolve throughout the entire software development process as more information is gathered or as the perception of existing requirements changes. In contrast to traditional views, requirements engineering is not a discrete, front-end activity of the software life cycle, but an ongoing process that starts at the beginning of a project and continues throughout the development process [12]. Requirements Engineering is a process that is commonly divided into five subcategories as defined by D. Zowghi:

- *Requirements elicitation* is the process of exploring, acquiring, and reifying user requirements through discussions with the problem owners, introspection, observation of the existing system, task analysis, and so on.
- *Requirements modeling* is the process of elaborating alternative models for the system; a conceptual model of the enterprise as seen by the system's eventual users is produced. This model is meant to capture as much of the semantics of the real world as possible and is used as the foundation for an abstract

description of the requirements.

- *Requirements specification* is the process of describing and formalizing model components to serve as a basis for contractual purposes between the problem owners and the developers.
- *Requirements validation* is the phase in which the specifications are evaluated and analyzed for correctness properties (such as completeness and consistency) and feasibility properties (such as cost and resources needed).
- *Requirements management* refers to the set of procedures that assist in maintaining the evolution of requirements throughout the development process. These include planning, traceability, impact assessment of changing requirements, and so on [12].

Good requirements engineering can make the difference between project success and project failure; it is an essential part of the initialization of development permeating the process.

Industry Perspective On Requirements Engineering

The diagnosis that a significant fraction of software projects waste resources, such as time and money, or even fail because of poor requirements engineering has created a demand for software developers with proficient requirements engineering skills. One of industry's major criticisms of current computer science education is that recent graduates lack these crucial skills.

Lack Of Management Skills

Requirements engineering is an interdisciplinary process; developers need to interface between a client's domain and the technical domain. An important part of the developer's job is to negotiate acceptable compromises between conflicting requirements, and to stay within budgetary, regulatory, and technical constraints [9]. A common source of problems in the software development process is that the fundamental assumptions of the system are never explicitly stated. An immediate consequence of this is the danger of faulty assumptions being used and propagated throughout the entire life cycle of the project [9].

When viewing the software development process from a requirements engineering perspective rather than from a technological standpoint, it becomes clear that software development includes much more than just programming. The failure to provide adequate integrated education in non-technical areas of software development, such as business communication and management skills, serves to validate industry's criticism of the curriculum.

Communication Problems

The elicitation of requirements is the delicate process whereby the developer captures the client's requirements [12]. This task is far from trivial; in fact it is the source of much confusion and the origin of numerous project failures. There are many reasons for miscommunications at this stage of development including lack of common vocabulary, poor interviewing techniques, lack of trust between participants, conflicts between participants, and uncooperativeness [10].

The developers and clients speak two different languages and operate with different sets of assumptions, which contributes to the formation of communication barriers. Communication is also impaired when developers fail to listen to their clients, take advantage of their client's expert knowledge of their own business processes, and do not encourage cooperation. [10].

The inexperienced developer often naively believes that the client is eager to assist in the requirements elicitation step. The requirements elicitation phase provides the clients with the opportunity to express their wishes, and explain their goals and business processes. The more experienced developer knows that client cooperation is not always the rule. The client often expects the developer to know the requirements without acquiring them and shows little interest in technology issues [2]. Some participants might also feel reluctant to aid the elicitation process because they feel threatened by the automation of their workplace. The degree of client cooperation is a reflection of the developers' mediating and interpersonal communication skills. Developers are responsible for making the client feel comfortable about sharing business processes and requirements.

A successful developer is a multi-talented individual that not only has proficient business management skills, but also exhibits strong interpersonal communication skills. The developer's job is to extract correct and complete system requirements from people with different roles, skills, and interests in the system, who may be unable or unwilling to clearly articulate their requirements.

A whole new spectrum of communication problems arises when shifting focus from conventional software development to global software development. The situation where different development teams and their clients are not physically co-located is getting more and more common in today's milieu of outsourcing. Research has shown that the most common means for communicating with other team members are informal discussions in hallways and lunchrooms that are impossible in global workspaces. Current research in communication and global software development is focusing on developing system requirements that all project participants can browse and update regardless of their physical location [3].

Attitude

Current research in requirements engineering indicates that there are considerable advantages that result from good requirements engineering, yet few organizations and few universities make use of it or teach it. The disturbing truth is that many still view requirements engineering as unproductive time. There is also evidence pointing to university faculty devitalizing the software engineering discipline and placing it below software technology. The result is that students can solve small well-defined problems but are unable to solve real world problems, which was one of the issues acknowledged by the software industry [6]. "It is well understood that a larger fraction of software projects fail than projects in any other branch of engineering. It is further understood that a prime reason for the high failure rate is the lack of acceptance of, even an aversion to, what is perceived by many software developers, particularly those with above-average technology skills, as unnecessarily harsh discipline required by [the] software development process [6]."

Other research projects indicate that this attitude is diminishing as requirements engineering pushes its way into the limelight. The recent attention given to requirements engineering is explained as a natural development from earlier advances in software engineering. Software engineers initially focused on programming methods and then on design methods. Requirements engineering now plays the lead role in the attempt to introduce more discipline into the development process [4].

Requirements Engineering In The Traditional Software Engineering Curriculum

The software engineering course at California State University, Long Beach (CSULB) represents what has become the traditional approach to software engineering education. Requirements engineering makes up only a minuscule part of this course. The class session that is devoted to this subject includes a brief lecture and a simulation exercise where students pose as developers and clients in an elicitation activity, according to Mr. Dennis Rice, part-time lecturer at CSULB, in an interview for this article. The main reason for spending so little time on requirements engineering, is the time pressure faculty feel to have students learn and complete the entire software development cycle in one semester. By only dedicating one lecture to the requirements engineering process, it inevitably becomes the discrete front-end activity industry warned about, where communication flows in one way from clients to developers. The results are requirements based on developers' interpretations of client's requests and requirements specifications and prototypes that lack clients' confirmation. Students have not been provided any real examples of client-developer communication or miscommunication prior to the elicitation process, and there is no check to see if the developers successfully captured clients' requests until the testing phase. Although Mr. Rice does not conform to the perception that requirements engineering is unproductive time, he is familiar with this notion both from his experiences working in the software industry and his experiences in the academic world. He feels that the focus on learning and teaching technical tools that are available to aid the development process, rather than teaching and learning the basic software development process, is unfortunate.

Closing The Gap

Fortunately, there is hope on the horizon in the form of those who believe in actively changing the status quo. Mr. Rice concludes that the best way to teach software engineering is to imitate the industrial software process. He also realizes that this is unachievable within the structure of the current curriculum. In his opinion, business communication should be introduced as early as possible into the curriculum, and the software engineering course should be divided into at least two courses to allow for the entire development process to be thoroughly covered. Another idea he supports is the introduction of software engineering elements, such as object oriented programming, modeling, requirements engineering, and testing in early programming courses to help mold student's problem solving approaches. Several ideas endorsed by Mr. Rice, along with the ideas of many others, have already been successfully implemented at various universities that have followed developments in software engineering and education and taken industry criticism to heart.

In the article "Incorporating the Client's Role in a Software Engineering Course [8]," Jennifer A. Polack-Wahl describes some of the successful changes she made to the second software engineering course for computer science, computer engineering, and information systems students. Prior to the changes, her students played the part of developers and determined all the necessary characteristics of the software they were developing. In order to prepare her students for situations more likely to occur in industrial software development, she allows them to experience the client's point of view. The students were divided into teams in which each development team was the client of another development team. At the conclusion of the semester the students accounted for their experiences as clients. Two out of three groups reported that they felt extremely frustrated when the developers did not listen to their requests. The frustration originated in feeling that the developers wasted time by developing prototypes that did not satisfy the requirements, a situation that could have been avoided had the developers listened to the clients from the beginning. Some software developing teams ignored the client's requests altogether and one development team justified it by telling the clients that their requirements were unnecessary. This resulted in clients mistrusting the developers and feeling that they had no control or influence over the development of the system that they requested. The relationship between one set of developers and clients deteriorated to the point that the clients refused to listen to any suggestions the developers made. Communication between the client and the developers was completely shut down because the developers failed to establish trust with their client. A common feeling among client groups was also that the developers did not prioritize understanding the client business.

Polack-Wahl describes the class experience as successful because "The entire class felt that the experience provided them with a better view of the reality of software engineering and that it would facilitate their future work with clients [8]". The students left the course feeling that clients should be regarded as experts in their field, and they grasped how valuable client knowledge is for designing an accurate and comprehensive system. They also got a taste of the multitude of communication obstacles that can arise between clients and developers [8]. Polack-Wahl's article characterized this particular course to be an advantageous way of learning software engineering. She suggested the introduction of this tactic in earlier software engineering courses, in order for students to utilize their communication and business skills in subsequent courses. Other universities have already discovered the benefits of introducing software engineering early in the education of computer science students, and have made software engineering a lower division requirement. By introducing software engineering principles early, they become the norm that students use when developing software, and are reinforced throughout the curriculum [5].

Future Directions

Few universities offer undergraduate degrees in software engineering [1], consequently, the majority of student's software engineering training is performed by computer science departments. Computer science departments, when instructing students in software engineering, need to address the criticism from the IT industry, evaluate their current curriculum, and make changes that have been tested and successfully incorporated. The focus of any changes must be on the software engineering discipline, as this is where students have the most to gain and where the industry has directed its criticism. Following is a list of suggested curriculum improvements that should be considered:

- Use project-based learning to improve problem-solving skills for real-world problems and to improve student's communications abilities.
- Emphasize the requirements engineering phase of the software development process.
- Distribute the coverage of software engineering over two courses.
- Integrate some of the software engineering elements throughout the entire computer science curriculum, starting with the introductory programming courses [5].
- Use techniques like collaborative learning to introduce students to communication, cooperation, critical thinking, and leadership competence within a technical scope [7].

Some computer science departments have strengthened their software engineering curriculum considerably by offering at least two courses. Given the many phases of software development, it is challenging for a faculty member to devote enough time on each of the phases in a single course. A distribution of the material also provides students with a more complete understanding of each phase and knowledge from the first course could then be applied and reinforced in the second course. These two courses would also allow time for a complete software implementation of the work students carry out in the software development phases. A suggested distribution of the material would have the students deliver a requirements specification document in the first course, which will then be implemented in the second course. During the implementation phase, the client-developer communication continues to evolve as prototypes are evaluated and confirmed.

The earlier students are introduced to the fundamental activities of the software development process, the smoother the assimilation of this body of knowledge will be.

This integration forms the foundation and outlook on software development that students can apply to the remaining computer science courses. An integration of this kind needs careful planning so that students experience a consistent coverage and so that the software engineering elements are enforced throughout.

Collaborative learning is not a central element in current curricula; most universities value and assess students on individual work. Too much focus on individualism in software engineering education contradicts the reality of the industry where software is developed through teamwork.

In summary, most of the techniques for improving software engineering skills in computer science students have originated from imitating processes used in the software industry. Curriculum changes that are motivated in this way can successfully address the industry's criticisms. The more closely software engineering courses resemble the reality faced by software developers in industry, the better prepared students will be to face that reality after graduation.

Conclusion

After investigating the software industry's academic demands and the universities' academic supply, the conclusion is that the industry has many valid criticisms of the current computer science curriculum. The universities fail to adequately prepare students for the real world because the development of many key skills is either partially or wholly ignored. Few courses focus on software engineering in general, and even fewer courses focus on essential specific skills such as client-developer communication. New graduates are ill-equipped to enter and survive a market with recessions, higher numbers of graduates, and outsourcing because they do not exhibit the qualities that the industry treasures.

Computer science and software engineering educators are now paying attention to the curriculum deficiencies pointed out by the industry. There are formal efforts to study these and other issues. In particular, a workshop titled "SEER: Charting a Roadmap for Software Engineering Education" will take place in March 2004 to identify the direction of software engineering education in the near future [1].

However, change will take time. In the mean time, students should supplement their academic knowledge from software engineering courses with some real-world experience. One opportunity that is available to many students is the ability to take an independent studies course. Such a course can be used to work on real-world projects from other university departments and from organizations or businesses in need of a software application. Students need to keep such opportunities in mind as they choose their courses. They should also demand that their schools provide the education they want and deserve - an education that will make them marketable and able to compete in the current business environment.

Acknowledgements

I would like to thank Dr. Rand Waltzman, Teknowledge Inc., for aiding my writing process with lingual and structural suggestions. I would also like to thank Mr. Dennis Rice, California State University, Long Beach (CSULB), for letting me interview him; his knowledge about software engineering helped me tremendously with this article. I would also like to give special thanks to Prof. Alvaro Monge, CSULB, for being my mentor by sharing his expertise and providing me with feedback when researching for and writing this article.

References

1

Bagert, D. J. "SEER: Charting a Roadmap for Software Engineering Education" *SEER*2 September, 2003. <**http://www.rose-hulman.edu/~bagert/seer/**> (6 November 2003)

2

Adaptable Process Model Software Engineering Checklist. R.S. Pressman & Associates, Inc. 2001.

3

Damian, D., Chisan, J., and Allen, P. *Awareness meets requirements management: awareness needs in global software development*.

4

"Introduction to Special Issue on Requirements Engineering". *SEI Interactive* March 1999 Volume 2. Issue 1. <**http://interactive.sei.cmu.edu/Features/1999/March/Introduction/Intro.mar99.htm**> (26 September 2003)

5

Jackson U., Manaris, B., McCauley, R. *Strategies for Effective Integration of Software Engineering Concepts and Techniques into the Undergraduate Computer Science Curriculum*. SIGSCE 1997.

6

Klappholz, D., Bernstein, L., and Port, D. *Tools of Outcomes Assessment of Education and Training in the Software Development Process* Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET 03). IEEE. 2003.

7

Koehn, E. *Assessment of Communications and Collaborative Learning in Civil Engineering Education*. Journal of Professional Issues in Engineering Education and Practice. October, 2001.

8

Polack-Wahl, J. A. *Incorporating the Client's Role in a Software Engineering Course*. SIGSCE 1999.

9

Sawyer, P., and Kotonya, G. *SWEBOK: Software Requirements Engineering Knowledge Area Description*. <**http://www.swebok.org**>

10

Scott, Clayton, and Gibson. *Knowledge Acquisition*. Addison--Wesley Publishing Company, Inc. 1991.

11

Warnke, T. "New Data on the IT Workforce Fuels Capitol Hill Debates on Immigration and Outsourcing". *Commission on Professionals in Science and Technology*, 17 September, 2003, <**http://www.cpst.org/ITWF_Press.htm**> (28 October 2003)

12

Zowghi, D. "Requirements Engineering Scope". 12 October 1995 <**http://www.jrcase.mq.edu.au/~didar/seweb/requirements.html**> (21 September 2003)

Biography

Kristina Winbladh (**kwinblad@cecs.csulb.edu**) is an international student from Sweden working on her undergraduate degree in computer science at CSULB. She started her studies at the Royal Institute of Technology in Stockholm before transferring to CSULB. Her future goals are to attain a MS degree and eventually a Ph. D. in computer science. She is interested in conducting software engineering research in the future.