SOFTWARE ENGINEERING REQUIREMENTS' ANALYSIS IN THE CLASSROOM

Nelson Luiz Passos
Department of Computer Science
Midwestern State University
Wichita Falls, TX 76308
passos@nexus.mwsu.edu

ABSTRACT

The computer industry evolution and the consequent complexity of software systems are requiring a better preparation of computer scientists with respect to software engineering. The recent shifts towards an accurate specification of the software system increases the pressure on the ability of a software engineer to understand a problem and correctly translate it in the initial design specifications. The classroom environment, however, is a controlled environment where, usually, the student is not exposed to the real problems that can be found in the industry. This paper reports a first experiment of teaching the students real-life problems. A methodology is proposed in order to develop a supporting tool for the delivery of classroom projects while maintaining a flavor of a real requirements' analysis project. The results of such an experiment show the initial weaknesses of the students in dealing with the unknown and the positive side effects of the method.

1. INTRODUCTION

The development of software systems is a complex task that requires a constant interaction between the end-user and the software engineer. Some instructors, with their skills developed in the academic environment, lack the necessary experience to transmit to the students the real problems found in the computer industry [3]. The emphasis being given to the accuracy of the software specifications, in order to improve productivity and to reduce errors, requires a better preparation of the software engineer in how to face the initial problem of understanding and specifying the target system. Since the existing analysis techniques are based on research efforts from academia, they will not reflect the real environment found in the field. This paper shows how a classroom method can be used to prepare the students for the surprises that are waiting for them in the industry, after graduation.

Most of the current research in Software Engineering considers the problems found in the development/operational process such as reusability, programming environments, reengineering, integration and test. Basili et al. [2], for example, study the concept of reusability in an object oriented programming environment. Sparks et al. [12] study the management of object reuse. Lanning and Khoshgoftaar [6] are concerned with finding a relationship between the source code complexity and its maintenance. Ward presents a transformational method for formal verification of theoretical specifications of complex algorithms [14]. All these studies emphasize the importance of Software Engineering. However, they do not contemplate the preparation of the professional that is needed to work with those techniques. In the
educational side of the Computer Science area, most of the studies focus on teaching methods related to the design and development of the software system. Jovalekic [5] shows an experimental work in applied Software Engineering where students were working on a set of given problems while taking the theoretical classes in every subject involved in the problem solving process. Several other experiments are related to teaching software engineering methods at all levels of the Computer Science curriculum, including initial freshman courses [13]. These topics are important in the overall context of the application of software engineering methods; however, they will fail if the initial analysis of the problem fails when done in a non-controlled environment.

A brief survey among some Universities has shown that most of the Software Engineering classes are focused on the development process of large systems under a well-controlled environment. Even the usual textbooks are biased when they describe the Software Engineering discipline as applicable to the development of large systems [8, 11]. In this study, we focused on small systems that, if not well specified, would cause as many errors as any other large system.

A regular undergraduate software engineering class was the target of the experiment. Instead of being submitted to the usual simple and intuitive problems described in the literature, the students were required to specify, design and implement a scientific algorithm, described in a technical paper. As we well know, technical papers are not the best source of information for describing a problem due to material that is left off during the publication, absence of practical examples, excess of theoretical text (including theorems, complexity analysis, etc.) which may not help in the final implementation. The use of the Waterfall model as the template for establishing milestones and follow-up reports made the material easy to control at the instruction level. However, the most important point to be evaluated was not the ability of the students in designing or implementing a system, but their skills in getting the remaining information that they need. Some surprising reactions at the undergraduate level were the initiative of the student in doing a literature survey in order to obtain more information. The position of the instructor in the project was to fulfill three different roles: the end-user, a technical consultant and the regular lecturer, providing general feedback to all students involved in the project.

The interesting, and exciting result, of this experiment was to observe the behavior of the students in getting involved with their end-user, almost totally ignoring the technical expertise that was available through the consultant, and their ability to manage their tasks in order to complete them on-time. Figure 1 shows the evolution of the interaction with the end-user while going through the first weeks of the project assignment. It is easy to notice a first natural reaction of the student trying to get the project done without any feedback from the end-user. However, the papers were chosen in order to require a strong interaction. Such necessity became clear when the project teams began to find out that they did not have the background to understand the problems assigned to them. This is what usually happens in the industry. In this study, we suggest a method to motivate the students to improve the interaction and at the same time to have a real life experience of industry problems.
In this paper we show how such a teaching methodology could be implemented using the current technology, the instructor's research background, and some motivational talks. We begin by showing in the next section some of the basic data available to the students and how they were assigned to their tasks. Section 3 discusses the initial results of the teaching method adopted and meaningful points not covered in this first experience, while Section 4 presents future plans on how to make the teaching work a little easier. A final section summarizes our observations on this effort.

2. BACKGROUND

The preparation of a Software Engineering class requires many tasks. The selection of a textbook that discusses most of the different topics of the discipline allows the instructor to present different approaches on how to develop a new software system. However, this instructional approach does not give the student any flavor of the real-world problems. Using the access provided through the Internet, a brief survey was conducted, in order to identify how other Universities with Computer Science programs were dealing with the Software Engineering classes.

Such a survey revealed that there are many programs that are addressing a specific software design methodology, such as Object Oriented, and do not cover all aspects of the Software Engineering process. Among those that are currently teaching the Software Engineering discipline on a regular basis, we selected some that had their class materials available through the Internet. For those selected, we
identified the class project assignments and how the students should accomplish them. Basically, we came up with projects being classified in three large groups:

1. **Fully specified problems** -- in this case, the students were assigned projects that were overspecified by the instructor, i.e., a full description of the system was given, and the students were expected to focus on the design and implementation phases of the project.

   A huge drawback of this system is the complete omission of the fundamental experience in how to initially investigate a problem and develop it from scratch.

2. **Double student role** -- in these assignments, the students' teams had two different roles, software engineers and end-users. As end-users, they should prepare the definition of a problem to be solved by another group. In such a role, they would act as customers and they were responsible for providing the data required by the group working in their proposed problem. As software engineers, they would face the problem defined by another group and go through the entire development process.

   This approach has a significant advantage over the first one since it gives to the students a better experience on how to conduct the entire process. However, the projects may go out of the control of the instructor, and will be restricted by the creativity of the team defining the problem.

3. **External end-users** -- in this situation, instructors opted for using external sources of information. This requires the student to go on field trips in order to identify the problem, and do the requirements' analysis. It was not clear whether the external sources would have any active participation in the process of providing information to the students or if they were just a passive element that the students should observe and then establish their requirements.

   Again, the main disadvantage becomes the lack of control of the developing environment. The result is a selection of projects that are easily identified and intuitively known by the students.

   With these three options at hand, the preparation of our class was not going in the desired direction, i.e., emphasize the initial phases of the system development process. This goal was established based on the assumption that many of the other stages of the software engineering process, such as implementation, test and documentation, are also covered in other courses.

**The experiment**

A first decision was made in order to motivate the students to work harder in the initial stages of the project. Five scientific papers, collected from recent conference proceedings and technical journals, were selected as possible project development goals. Such papers presented complex scheduling algorithms, which could be used by compiler optimizers, as the problems to be implemented following a very well specified development plan. The students were teamed into four groups and they were called to choose the paper that they would try to implement in a drawing process. At this point the students had no idea of the contents of the papers
or which one they would be assigned. The four papers selected [4,7,9,10] were
distributed and the first student reaction was, as expected, negative: "...the papers
are unreadable...". This was the desired feedback: it implied that the project was not
of intuitive solution and would require a substantial effort in obtaining the correct
definition and specification.

After the problems have been assigned, each team was required to choose
specific functions for its members. The students assumed positions as managers,
analysts, developers, integrators and sales representatives. Under this organization,
their responsibilities included:

- **managers** - schedule and report project milestones. Also responsible for reporting
  project delays, obtaining deadline extensions, and reporting project problems.

- **analysts** - problem definition, requirements' analysis, system specification, and
  sales support.

- **developers** - software design and implementation.

- **integrators** - test specification, system verification and operation.

- **sales representatives** - project proposal, presentations, prototyping, and final
delivery.

This idea of assigning specific responsibilities to the team members is not
new. However, the inclusion of a sales element reinforced the idea of strong contact
with the end-user in order to "sell" their design.

Figure 1 shows how the students reacted to this approach by measuring the
effective contact of the team members with the instructor, in this case in the role of
end-user (without a strong knowledge of the product). Just in case, the instructor was
also available to the students as a technical consultant. The initial contact between
the students and the end-user as expected was minimum. After the initial "shock"
caused by the papers, the groups began to improve their participation by probing the
end-user as much as possible in order to write their system specification. An
important difference between this idea and the overspecification of the assignment is
that, in this case, the students got only the information that they asked for. The
improvement on the instructor-students contact resulted in a well-controlled
development environment, where the instructor was able to assess the knowledge of
the students and concurrently, make the adjustments in the class material in order to
provide an effective feedback.

A first observation of this methodology is the fact that the students, in a
regular Computer Science program, lack preparation on how to face the real
problems that they can find in the industry, including human relations. The textbook
by Sommerville covers this topic [11]. However, as a part of a general Software
Engineering course it is more probable that such a chapter in the book is going to be
skipped.
Current teaching techniques are restricted to the analysis and specification of business applications which may be intuitively recognized by the average student who is going to deal with the problem. Such situation causes a reduction in the interaction process since the student does not depend on the instructor’s information. This also gives to the student the wrong idea that they will be able to dominate the software development environment.

3. FIRST RESULTS

In this section we discuss some of the observations of our experience teaching the student how difficult it is to find the correct solution for a real life problem. Some not expected “side effects” of this method included a sudden interest of some undergraduate students in conducting individual research and preparing technical papers for review at regular conferences or symposiums.

Other interesting and targeted results reflect the expected shift of the work effort to the first stages of the project. It was well known that the algorithms described in the papers provided to the groups would not require a huge effort of coding; however, they would be difficult to understand. Therefore, the intense interaction that occurred in the specification process made it longer than usual and a real example of what is expected in a well developed system.

After we detected that the experience was producing the expected results, we began to analyze what was missing. Any instructor with some experience in the industry would know that one point not covered in this process was how the behavior or personality of the end-user would influence the analysis of the problem. In general, textbooks and instructors show the end-user as an element avid for getting the system developed, totally cooperative and collaborative. This is not really true. In the real world, some of the end-users do not participate in the decision process; however, they are called upon to provide the necessary data for the system analysis. In this case, many of these individuals will become a tough obstacle in the system development process. Several factors may affect the communication between the software engineer and the end-user. If we ignore the environment conditions and consider only the basic personality of the end-user, then studies have shown that individuals can be qualified in three groups according to their personalities [1]:

1. Task oriented (also called detached), where the end-user expects to receive clear technically sound explanations of decisions taken during the specification process.

2. Self oriented (also called dominant), when the end-user would like to have the system developed according to his/her specifications and is not willing to help the software engineer in the development process.

3. Interaction oriented (also called dependent), where the end-user feels comfortable in working with a team and will accept any decision made by other team members such as the software engineer.
Figure 2 shows the role that the software engineer should assume in order to overcome those end-user personality characteristics. As we see, the software engineer should adapt his/her own personality to the end-user behavior. Therefore, he/she needs to be a good communicator as well as a good listener, balancing those qualities according to the user personality.

In our experiment, the instructor assumed the position of task oriented end-user to assure a good technical feedback for the students. Such a position should be expected, anyway, from an instructor. The development of alternate situations must be studied and applied in order to complement the real development environment in the classroom. A possible way to introduce this extra parameter in this process is described in the next section.

4. THE REAL (VIRTUAL) USER

As it is easy to notice, the amount of work required from the instructor to follow on with this practice could become excessively high, mainly in small institutions more teaching oriented. Additionally to this work, we should also consider that the projects being developed simultaneously by the students are not on the same topic. Also, for a successful implementation of the course, the instructor should have the necessary skills to show different personalities when in the end-user role.
The solution to this problem resides in the new technologies that are now available to any computer literate. The use of multimedia applications seems to be the most appropriate solution. A software tool for computer based education development is now under evaluation to verify if it can fulfill this role [15]. The ability of combining multimedia with computer based education will allow the development of artificial end-users which would provide the problem information to the students according to different personalities. Different project groups would be submitted to different projects and users. The implementation of such techniques is not too complex when using the CBE tool. Figure 3 shows the expected image of a “dominant” end user explaining the development of a sale monitoring system. Even considering the availability of audio features in the software, a first implementation would be done with balloons (as in comic strips) in order to avoid investments in new equipment. Buttons in the screen (or hot spots) would allow the student to obtain more detailed information as well as to be evaluated in his understanding of the problem. The implementation of such a technique will reduce the presence of the instructor in the specification process, while maintaining the controlled environment necessary for a good observation of the learning process.

5. SUMMARY

The complexity of software systems requires a better preparation of computer scientists with respect to software engineering. The preparation on how to understand a problem and correctly translate it in the initial design specifications becomes an essential factor in the formation of a software engineer. The classroom environment, however, is a controlled environment where, usually, the student is not exposed to the real problems. This paper presented a first experiment in challenging the students to solve problems that are not intuitively recognized and the first results in achieving a better understanding of the development process. The methodology proposed gives a flavor of a real requirements' analysis project while maintaining a controlled study environment. The results of such an experiment show an initial weakness of the students in dealing with the unknown and some positive side effects of the method. Future developments require a long term investment in the development of multimedia tools that will support the class work.
6. REFERENCES


