Abstract—The majority of undergraduate software engineering courses are taught by presenting theoretical background, reinforced by group or individual work on in-class projects. However, exposure to real-world projects would greatly improve a student’s performance in industry. So far, attempts at involving students in real-world projects have been focused on large software products. This approach almost never succeeds in producing a final workable product because of the short period and provides little satisfaction to the customer or to the students. Hence, a fundamental question arises: can we develop a one-semester undergraduate software engineering course where we teach software engineering concepts and methodologies, and at the same time, have each student experience every software engineering phase, work in teams on a real-world project, interact with a real-world customer, and develop a marketable product which can be used, as is, by the customer? In this paper we answer this question in the affirmative by presenting a novel approach for teaching software engineering to undergraduates.

Index Terms—Entrepreneurship, Real-world project, Software engineering education, Undergraduate

I. INTRODUCTION

The Computer Science and Engineering curriculum usually provides room for only one mandatory software engineering undergraduate course. The majority of undergraduate software engineering courses are taught by presenting the concepts and methodologies and by assigning either group or individual in-class projects. More innovative approaches [3], [12], simulate the industry working environment by having the entire class work on a single project with groups of students assigned different tasks within the “organization”. While this approach resembles the industry practice in research labs and medium-size companies, the fact that each group of students focuses and practices only on a specific set of skills may restrict students’ opportunities after graduation. We believe it is better for each student to practice all software engineering phases.

Exposure to real-world projects, derived from the needs of real-world customers, would greatly improve a student’s performance in industry. So far, attempts at involving students in real-world projects [2], [4], [5], [6], [7], have been focused on large products, with the intent to give the students the opportunity to exercise their software engineering skills on projects that closely resemble the industry environment. Unfortunately, this approach almost never succeeds in producing a final workable product due to the limited time available and provides little satisfaction to the customer or to the students. This approach has the potential to be more successful if a two, or even three, software engineering course-sequence finds its place as a requirement in the curriculum.

In this paper we present a novel approach for teaching software engineering to undergraduates within a semester-long course. This course incorporates the study of software engineering concepts and methodologies, while allowing each student to experience every software engineering phase. Students work in teams on a real-world project, interact with a real-world customer, and, in the end, succeed in developing a marketable software product which can be used, as is, by the customer. We also present two successful student stories based on the outcomes of Spring and Fall 2004 semesters. The projects were suggested by Glassboro Economic Development Corporation (GEDC). The Spring 2004 project, Gap Financing, is currently being used by GEDC to attract businesses to the Glassboro community. The Fall 2004 project, Assistance Program, is used by GEDC to provide a simple way for community residents to determine eligibility for grants GEDC has available.

The rest of the paper is organized as follows. We present our approach in detail in Section II. In Section III we present two successful student stories based on the outcomes of two semesters, and provide background on the two projects. In Section IV we present the student reaction to our approach, with minimal input from the course instructor. Finally, we conclude with a summary of major contributions in Section V.

II. APPROACH SETUP

Software Engineering is typically an upper-level undergraduate course in the Computer Science and Engineering curriculum. Most students enrolled in the course already have a strong background in programming. However, students generally have not developed appreciation for careful planning and have not produced the extensive and comprehensive documentation related to real-world projects. The majority of the time their assigned projects had clear descriptions, with clear grading schemes, leaving little room for interpretation. If this was not the case, students developed requirements using a build-and-fix approach. Their experience working in teams may also be limited.

The root of our approach is the belief that the best way of learning is by doing. The best way of doing is by actively participating in real-world projects, for real-world customers. Indeed, students are inexperienced using software engineering techniques, which they are concurrently learning, but if they...
are properly guided, they have the potential to successfully apply these techniques in a real-world setting.

A. Instructor's Role

Locating an appropriate software engineering project requires careful selection by the instructor. In our approach, the instructor has the following responsibilities:

- **Identify an appropriate project.** A key to our approach is to find a real-world customer that has a small-enough project for the students to complete within one semester, and at the same time, be valuable to the customer. Hence, the selection of the project must be carefully planned by the instructor prior to the beginning of the semester. The instructor must contact potential customers to identify general requirements for their projects. Once the main features of a project are clearly identified, the instructor, based on experience, decides whether the project is appropriate. Also, the instructor evaluates the potential marketability of the project.

- **Provide software engineering background and sample documents.** Once the semester starts, the instructor lectures about all aspects of modern software engineering. Since most software projects fail due to lack of time, the instructor must provide detailed lecture notes and interactive lectures, to minimize students’ learning curve and hence maximize effective time they spend on the project. In addition, the instructor must provide sample documents for each software lifecycle phase. This greatly helps the students to structure their documents in a professional manner.

- **Divide the class into teams.** All teams go through the entire software lifecycle by working separately on the same project. The class is divided into teams of four or five students. Other approaches [5], have used teams of seven or eight students, with the intent of achieving more in a shorter period. Dividing the class into such large teams would work if all students were able to be together, as in a working environment. However, in practice students have completely different and busy schedules which makes coordinating outside-of-class team meetings nearly impossible. This may lead to situations in which students from the same team work on the same part of the project, resulting in non-efficient use of time. With teams comprised of four or five members, outside-of-class team meetings are easier to schedule, and proper communication and coordination are easier to achieve. The students complete a questionnaire at the beginning of the course. These questionnaires are used to assemble balanced teams based on the students’ backgrounds and areas of proficiency. The instructor’s aim is to assemble teams of complementary members to mimic those found in small start-up software businesses or in small-scale development groups in industry.

- **Distribute the initial requirements.** In order to provide a realistic environment, the customer alone develops the initial requirements for the project. In reality, these initial requirements are often vague, especially if the customer does not have a well-developed technical background. The instructor distributes these requirements to the students.

- **Coordinate the initial meeting between the customer and the students.** The instructor organizes an open forum meeting (usually during class-time), where the customer can answer questions from all teams. Additional meetings are scheduled between the customer and each team.

- **Act as overall project manager.** The instructor assists the teams’ development process in a light-handed way to avoid any standardizing influence and not squelch the creativity of the students. Students have full control over the development of the software, such as choice of architecture, module design, or programming language. Upon completion of each phase, each team receives a critical analysis of their deliverables. The instructor also makes final judgments in situations of teams’ fire and hire decisions (see Subsection II-D).

- **Assign final grades.** The instructor assigns the grades independent of the customer’s assessment and opinions.

B. Students’ Role

It is important for the students to play an active role when participating in a software engineering project, since they will be applying the techniques they learn during lectures. In our approach, each team is expected to:

- Initiate a professional relationship with the customer.
- Produce a set of questions based on the initial project description.
- Submit weekly team logs to allow the project manager (i.e. the instructor) to follow their progress.
- Perform requirements analysis and specification; immerse themselves in the customer’s working environment.
- Design the modules and the architecture of the system.
- Choose an appropriate programming language and implement the design.
- Perform structural and specification-based testing.
- Produce supporting documentation.
- Prepare a professional customer-oriented presentation (agenda, brochure, poster, demo of the final software implementation).

C. Customer’s Role

In our approach, it is the customer’s responsibility to serve as a competition advocate. Though the expectation of success on behalf of all teams cannot be expected, the customer must engage teams with equal efficacy. This prevents a team from acquiring improper advantages due to the customer’s influence. The customer is expected to:

- Develop the initial requirements of the project.
- Time requirements of the customer may be difficult to plan due to the number of teams and same project schedule but different academic schedules. The customer must, therefore, negotiate availability with teams during the requirements analysis and evaluation phases. The
customer must set aside an average of four hours for interaction with each team for the entire semester.

- Act as liaison between the students and the users of the system, during the requirements analysis phase.
- Organize the awards ceremony (see Section II-D), and play a key role in the marketability of the product.
- Choose a winning team and place the other teams in order of success, based on how well each team’s software product meets customer’s needs.

D. Incentives for Students

Learning software engineering techniques and applying them on a real-world project is, by itself, a motivating factor for the students. Nevertheless, further motivation can only inspire students to perform at a higher level.

In our approach we use the following additional incentives:

- **Invaluable Experience.** Exposure to real-world software engineering projects gives students real-world experience employers seek.
- **Giving Back to the Community.** The students have the opportunity to use skills they have gained in class for the benefit of the community.
- **Team Competition.** Teams compete to have their software product chosen by the customer as the winning product. Competition induces teams to negotiate adding requirements and features, instead of removing requirements and features. This realistically simulates the real-world environment when small companies compete for projects.
- **Awards.** The winning team gets the most benefits for their efforts; second and third place, as well as honorable mention certificates, are awarded to the rest of the teams. The prizes are awarded during the awards ceremony organized by the customer at the end of the semester.
- **Publicity.** The customer and the instructor choose projects with strong publicity potential and make efforts to publicize the students’ work.
- **Monetary Incentives.** Teams have the opportunity to market their software product to similar businesses.
- **Advertising and Marketing Support.** The customer provides free advertising to the winning team.
- **Test-bed for Future Versions.** The customer receives free software upgrades from the winning team, and at the same time, the students are able to test their products before releasing them to corporations with whom they have contracts.
- **Fire and Hire.** The main problem with teamwork in a college environment is that not all teammates participate at the same level [8], [13]. To avoid having students “carried” by their teammates, we established a proviso that each team takes steps to fire one or more teammates, in extreme cases. A team gets the chance to develop a "case" against a particular team member who does not perform his/her duties. The "case" is then brought to the instructor’s attention, who first tries to mediate the conflict, and if does not succeed, the offending team member will be expelled from the team. In general, this means that the student fails the course. Teams could also hire students from other teams. A team that feels inadequately competitive may hire, with approval of the instructor, a member of another team. This proviso emphasizes a real-world atmosphere and encourages productivity and a quality effort from each team member.

E. Approach Limitations

The main tradeoff of our approach is that, by choice, students work on small-scale projects with the goal of completing them, thereby avoiding challenges encountered in large-scale projects. In reality, as mentioned in Section I, such large-scale projects have proven difficult to accommodate in an educational setting. Our approach offers most of the challenges software engineers face, including the full lifecycle of a robust, albeit small-scale, software development project, including the need to establish a design to accommodate the future.

Nowadays, it is increasingly common for software to be developed by extending and modifying existing systems. In our approach we focused on developing software systems from scratch. We believe the same technique can be adopted and used in upgrading existing systems.

In order to shorten the project’s delivery time, authors [1], [2], suggest the use of advanced CASE tools, such as automatic code generation tools. We believe that novice software engineers should not rely on esoteric technologies and CASE tools, in order to first gain understanding of the need and place for these tools, which can result in their use at full capability.

In our approach, the instructor assumes the role as project manager. We believe full transfer of project management responsibilities to students would be difficult, due to students’ inexperience. We plan to increase students’ exposure to project management in future offerings, and analyze the effects on their performance.

III. Background for the Projects

In Spring and Fall 2004 semesters, our students developed two software products for the Glassboro Economic Development Corporation (GEDC): **Gap Financing System** is being used to attract businesses to the area, and **Assistance Program System** is being used to provide community residents with accurate information about grants for which they are eligible.

A collaboration between a University and the local Economic Development Corporation is a win-win situation. The students benefit by learning the software development process and beginning to work with real-world customers. In addition, as also established in [5], the students are more motivated in working on socially-relevant projects. The local community development corporation benefits from receiving free software which has been designed to suit their needs.

In the past, GEDC used paper-based systems together with a diverse set of commercially available software tools to handle their workload. Two of the inefficiencies were having to manually input data multiple times and having the burden of opening and closing all the various all-purpose tools. Therefore the tasks were to integrate the functionality of these discrete
packages into virtually seamless tools, that could quickly and efficiently produce complete reports.

A. Gap Financing Project

Businesses contemplating relocation often contact local Economic Development Corporations which have inventories of local assets, such as: tax abatements, available plants or real estate, free training provided by local schools, associations with local industries, etc. EDCs from different communities compete to attract businesses and bidding wars typically ensue. The proposals generated in response to each individual query are referred to by GEDC as Gap Financing. They are referring to all the advantages they offer as means to fill any remaining economic gap needed to convince the prospective business to relocate to their community.

It had taken GEDC almost two weeks to produce all the needed scenarios for one business. With the new software, they produce these scenarios right on the spot.

In order to develop this software product, the teams were required to provide a user-centered interface design, and respond with custom reports and visual aids comparisons for known alternatives (i.e. buy, rent, lease, multiple finance rates, etc.). Additionally, accommodations had to be made for the archival of data to support multiple database systems and future enhancements such as networking.

B. Assistance Program Project

GEDC provides assistance to the local residential population through several programs such as the Neighborhood Preservation Program (NPP). This program provides grants to home and business owners in an effort to help improve their residence in an overall attempt to improve the quality of life in the community. Residents and business owners located in a specific target area may be eligible to apply for NPP grants.

With the new software, GEDC determines an applicant’s grant eligibility in minutes. A user-centered interface design allows an administrator to easily modify existing grants’ eligibility rules, add new grants and rules, and add/edit the information regarding contractors performing the needed rehabilitation work. To simplify the submission of information to applicants, contractors, and the New Jersey Department of Community Affairs, the software also automatically generates reports and visual aids.

C. Projects’ Success

With the help of GEDC, which advertises the Gap Financing System during national conventions, the Spring 2004 winning team is currently marketing their product to other EDCs in the USA. Since the system was developed to be easy to change, the students can quickly adapt it to specific needs of other EDCs.

The Assistance Program System has attracted interest from the New Jersey Department of Community Affairs, which sent a representative to attend students’ final presentations. The state of New Jersey has made the decision to fund the state-wide adoption of the Assistance Program System.

Students have received gratification by having their work publicized in local publications such as Plain Dealer and Rowan Magazine (the University’s alumni magazine), for the Gap Financing System, and Gloucester County Times, for the Assistance Program System.

The customer has expressed their satisfaction with the result of the collaboration. The director of GEDC declared the following in an article posted on the University’s website:

“We are excited and honored to have been able to work with (the) students on this project. The students have a valuable software package that will assist our office and potentially hundreds of others across the state in administering grant funds to low- and moderate-income homeowners. The software will be implemented and used to make our office many times more efficient than it already is, and in exchange (the) students have had the valuable chance to gain real-world experience.”

IV. Students’ Experiences

Five students who have taken the course were chosen from different teams and different semesters to provide a broader spectrum of experiences. In order to realistically capture their perspective on the approach, this Section has minimal input from the instructor.

Table I shows the assigned schedule for each software engineering phase, along with the average actual time consumed. The excitement of a real-world project and the competitiveness aspect of the course dramatically increased our level of motivation early on. At the beginning we learned the significance of choosing the appropriate lifecycle model for the job. Due to the importance of providing a user-friendly graphical user interface, the lifecycle model chosen by the instructor was Rapid Prototyping.

A. Requirements Analysis and Specification

Our instructor and the customer agreed that each team should have private meetings with a GEDC representative and with the future system’s users. The intent of these private meetings was to ensure that we practice our interview skills learned in class in a real-world setting, and in addition, any proprietary advantages a team invents remain with the originating team. Therefore, the requirements were not determined by the instructor and simply given to the students. Instead, the requirements analysis was exercised by each team through its interaction with the customer and with the users of the system.

<table>
<thead>
<tr>
<th>Project Phase</th>
<th>Planned Interval</th>
<th>Actual Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Analysis</td>
<td>2.5 weeks</td>
<td>2.5 weeks</td>
</tr>
<tr>
<td>Requirements Specification</td>
<td>1.5 weeks</td>
<td>1.5 weeks</td>
</tr>
<tr>
<td>Design</td>
<td>4 weeks</td>
<td>4 weeks</td>
</tr>
<tr>
<td>Implementation</td>
<td>4 weeks</td>
<td>5 weeks</td>
</tr>
<tr>
<td>Testing</td>
<td>2 weeks</td>
<td>1 week</td>
</tr>
</tbody>
</table>

October 19 – 22, 2005, Indianapolis, IN

Session FIG

35th ASEE/IEEE Frontiers in Education Conference

FIG-27
During the process of developing the system’s requirements, our goal was to correctly interpret the needs of the customer. In order to better understand these needs, each team had to research the existing systems, analyze the users’ day-to-day work, and produce various scenarios based on their findings. These activities allowed us establish the functional requirements and organize system’s software qualities in order of their priorities. At the same time, the lack of a technical background on the part of the customer and inexperience on our part revealed the difficulties of unearthing customer’s needs and designing a prototype to match users’ requirements. Changes made by the customer and by the users during the requirements analysis process forced us to cope with adversities not encountered in the normal academic environment. We realized that software engineering projects are open-ended, which is contrary to typical in-class projects. New topics for us were the evaluation of future changes, the discovery of potential risks, and the identification of non-functional requirements. Considering potential risks, future changes, and non-functional requirements was important, as it aided in the production of acceptable software. Additionally, a challenge was to negotiate away, in a professional manner, by not offending the customer, constraints which limit implementation freedom and requirements which are not technically feasible. Ultimately, we learned how to transform a relatively vague objective into a clear, concise description of the software requirements.

B. Design

The design phase of the software engineering process was new to most of us. A greater appreciation for the requirements specification was gained in this phase. This was because without a firm understanding of the requirements, the design would have been corrupted and lacking in sufficient quality. For instance, without proper identification of all future changes, our design could not absorb them easily, thus making our software product less valuable to the customer. We noticed the helpfulness of the architectural design and of other diagrams in conveying the structure and the functionality of our system. More importantly, we realized the difficulties involved in creating a design which meets all the specifications. We learned there are many ways to get the job done and finding the best solution may not always be a clear decision.

Our biggest design task was to decompose the system into modules with high cohesion and low coupling, which would allow the modules to function with minimal exchange of information. For each module, its purpose was developed to assist in the understanding of what the module would do and its rationale was stated to present why the module was needed. Also, included in the design of each module were its provided and required interfaces, which defined the relationships among all the modules as a collective system. This structure permitted us to work on individual modules separately, and thereby increased our team efficiency.

Despite the fact that the teams were given full flexibility and control, sample design documents authored by students who already completed this course, were provided for our reference. The samples were especially helpful in the development of the design because all teams used the format as a template. Lectures, classroom discussions, and examinations further aided the understanding of the design concepts and methodologies.

C. Implementation

During the implementation phase we gained greater appreciation for the design phase. We found it easier to code having a detailed software design for reference. Most teams had to meet often in this phase to bridge various modules together. We found that putting time and effort in the design phase allowed us to understand the parts necessary to build the software and facilitated the implementation process. We realized that learning new material quickly was indispensable in keeping the project on schedule. Hence, team members were assigned to research unfamiliar domains. While one team member was researching new information, the others continued working in parallel. This helped productivity and created a solid team environment.

However, the implementation phase was not without its difficulties. This included choosing a suitable programming language, overcoming technical challenges, dealing with conflicting time schedules, and accepting new ideas. For instance, the project required a programming language that could be used to create user-friendly graphical user interfaces. Some students were not proficient in the programming language chosen by their team, thus the process of mastering a programming language accompanied this phase. Human interaction skills played a huge role in keeping the teams focused.

D. Testing

Our previous experiences with testing had been limited to boundary testing, and formal methods had never been emphasized. We employed the new techniques of testing learned in class which boosted our confidence in our product. This included having team members review code and develop structural and specification-based test cases for modules not written by them to avoid blind spots and hind sight. The assertions developed during the implementation phase proved to be very helpful in correcting improper assumptions between modules, and allowed us to successfully perform all integration tests developed during the design phase. Finally, we performed the acceptance tests using real data provided by the customer during the requirements analysis phase. Testing phase exposed the commitment necessary to develop quality software.

E. General Lessons Learned

In addition to lessons learned in each software engineering phase, general lessons were also learned. These include the following:

- **Entrepreneurship.** Although there was only one winning team, many of us left the course feeling we were all winners. Despite our short exposure to the process, we now feel we have better knowledge of developing high quality software. Some teams welcomed the idea of
working together and starting a business in software development. Prior to this class, many of us had not contemplated software development as a possible avenue to starting a business. Some of us, including those in non-winning teams, have been approached by local businesses and organizations to develop custom software. These organizations have expressed similar needs, such as replacing their paper files with databases and automation.

- **Rejection.** For most of us, rejection was a new obstacle. Learning to accept new ideas and criticism was not easy. We had to learn to adapt our ideas to the needs of the customer. There were times when we had to make major changes to both the design and implementation. Despite some initial misgivings and shock, we knew that we were ultimately working towards a better product.

- **Importance of Research.** We learned the importance of researching details before embarking in a new technical domain. This was notably true in this experience based on the majority feeling that if research had been completed, the result would be improvement in project performance.

- **Importance of Good Team Dynamics.** Ideally, teams organize and devise methods to distribute the workload evenly. In some cases, certain members took ownership of the project, a phenomenon known as project hijacking. This emphasized the importance of having a team with good dynamics.

- **Effects of Teams’ Displacement.** Due to the hire and fire mechanism, members displaced from their original team learned the effects of team efforts (good and bad). New team members suffered bruised egos in having to accept existing development status, such as choice of system architecture or implementation language.

- **Real-World Deadlines.** The majority of us had been exposed only to in-class programming assignments previous to entering this course, where an incomplete assignment receives partial credit. In real-world, incomplete projects merit little credit. Only when time becomes short and deadlines approach, everyone realizes the significance of communication, coherent requirements outlines, and stable interface definitions.

- **Human Factors Engineering.** Conveying technical information to a non-technical audience proved to be a challenge. We learned to make specific efforts to carefully review all user interfaces to ensure they are intuitive and rational. These efforts included developing a context-sensitive help system and wording positive error messages tailored to the users’ skills.

- **Professionalism.** Overall, we learned first-hand that we can be competitive in the real-world only by presenting ourselves at a high level of professionalism in our relationship with the customer, and by ensuring that our work meets the highest standards possible.

V. CONCLUSION

In this paper we have presented a novel real-world approach for teaching software engineering to undergraduate students within a one-semester course. Our approach previews the forthcoming challenges students will face in industry and ignites their entrepreneurial spirit. Students were challenged with deadlines and responsibilities to perform at a high level. They discovered that time management, careful planning and monitoring, and timely and thorough coordination are crucial for software engineers. Awards, marketing, and professional recognition served as enticements forming cohesive bonds between teammates and strengthening each team’s desire to succeed. All students benefited from this maturing experience and, with their broadened perspective and keener insights, are better equipped to deal with any phase of software development, as well as with basic aspects of project management.

Two concurrent paths need to be followed to successfully complete a task of this nature, one path being a real-world project and the other being the study of relevant software engineering topics in textbooks [9], [10], [11]. The overall result of this experience is the most satisfying example of a win-win experience, for the students primarily, but also for the University and the community.

ACKNOWLEDGMENT

We are grateful to Mr. Sam Leone, Director of Glassboro Economic Development Corporation of New Jersey, for supplying real-world projects, his time and trust in us.

REFERENCES


