On a Pursuit for Perfecting an Undergraduate Requirements Engineering Course

Chandan R. Rupakheti, Mark Hays, Sriram Mohan, Stephen Chenoweth, and Amanda Stouder
Department of Computer Science and Software Engineering
Rose-Hulman Institute of Technology
Terre Haute, Indiana 47803
{rupakhet,hays,mohan,chenowet,stouder}@rose-hulman.edu

***Abstract*—Requirements Engineering (RE) is an essential component of any software development cycle. Understanding and satisfying stakeholder needs and wants is the difference between the success and failure of a product. However, RE is often perceived as a "soft" skill by our students and is often ignored by students who prioritize the learning of coding, testing, and algorithmic thinking. This view contrasts with the industry, where "soft" skills are instead valued equal to any other engineering ability. A key challenge in teaching RE is that students who are accustomed to technical work have a hard time relating to something that is non-technical. Furthermore, students are rarely afforded the opportunity to practice requirements elicitation and management skills in a meaningful way while learning the RE concepts as an adjunct to other content. At Rose-Hulman, we have experimented with several project-based approaches to teaching RE, which have evolved over time. In this paper, we document the progress of our teaching methodologies, capture the pros and cons of these varied approaches, and reflect on what worked and what did not in teaching RE to undergraduate engineering students.**

***Index Terms*—Requirements Engineering; Project-Based Learning; Course Evolution**

I. Introduction

It is well known that engineering requirements is not a simple task. To prepare our students for a successful software engineering career, we strongly emphasize requirements engineering in our curriculum at Rose-Hulman Institute of Technology. In this paper, we describe the development of a junior year course in Requirements Engineering, which is required for both Computer Science (CS) and Software Engineering (SE) majors.

The need of such a course is supported by the philosophy of *impedance matching* undergraduate student learning with their entry and exit points. We generally agree that incoming students do better if they recognize college as an extension of their learning in high school. Similarly, most of us offer a capstone course in the senior year to ease students' entry into professional life after graduation. For the more specifically targeted engineering schools, both first-year attrition and job fit at graduation are special concerns. While we have additional program goals like inspiring students to pursue engineering work, and improving professional practices via delivering students with new capabilities, the systems concept of matching students as inputs and outputs continues to have face validity in curriculum decisions. At Rose-Hulman, 90% of our CS and SE majors go straight into software development work, so preparation for this career is our prime mission.

Leaders in the software industry sport slogans like, "We hire for technical skills but promote for people skills." Indeed, recruiting organizations may hire for the former largely because differences there are easier to detect in interviews. Computer science departments, in contrast, are clearly prejudiced against the value of "soft skills," often relegating all teaching of these to humanities departments. This bias is passed along to students, who tend to acquire a sense that only technical abilities have inherent significance.

At Rose-Hulman, courses in our CS and SE majors are experiential, and our students believe they "know something" when they can do it. Thus, when we invented a course in Requirements Engineering, we felt students had to do work which simulated real world settings. This course has been updated, from 2003 onward, to reflect both our own experience teaching the subject and the research support for courses of this type. For example, we emphasize the recommendations of Macaulay and Mylopoulos that requirements be seen to be conflicting and changeable, and that justification and traceability be insisted upon as arbiters [9]. In Section II, we summarize current research in requirements education.

Table I highlights the stages of our course development. The reader can see that, during different eras, we alternated between having students tackle larger or smaller requirements elicitation projects. The crucial dimension of "where the client came from" also varied over time. For example, in 2015 we allowed students to form their own companies and be entrepreneurs, quite a distance from having instructors or actors role-play the client, or from using real clients who wanted a real project done. In 2011-2014 our "Integrated" team formation had entire classes of 20 or more working on the same project, divided into sub-teams.

In Section III, we explain the thought that went into each of these varying approaches, the basic pedagogical mechanisms used, and the pros and cons of the outcomes. Section IV does a comparison using data collected at the time the classes were taught. Section V revisits the validity of our results, and Section VI concludes the paper.

II. Related Work

In this section, we describe studies backing both the current format of our course and in support of earlier versions of the

2377-570X/17 $31.00 © 2017 IEEE
DOI 10.1109/CSEET.2017.24

TABLE I Overview of Course Projects.

Year	Feature Size	Client Status	Team Formation
2003-2006	Reduced # of features, 3-6	Role Play	All Students - Same Project
2007-2011	Reduced # of features, 6-8	Real Client	Varied
2011-2014	Large # of features, 20	Real Client	Integrated
2015	Reduced # of features, 3-6	Entrepreneurial Minded Learning	Varied
2016 - Present	Large # of Features, 20-30	Instructor Controlled Role Play	Varied

course. Many resources now exist for case studies emphasizing requirements issues [5], [7], [8], [15], [20].

Leffingwell and Widrig present a framework for eliciting software requirements. They provided a template titled "Context-Free Interview" for constructing a 1-1 interview with any stakeholder. Each section in the template has an iterative exploration of the client's problems with set stopping criteria. We use this Context-Free Interview to frame elicitation as an algorithm, not an ad-hoc conversation [8].

Woolcock described his peer-pressure tactic for building a course with a required reading and discussion component. At the start of each class, he breaks up the students into randomly assigned small discussion groups. When he solicits whole-group discussion, he asks each group sequentially to provide commentary. Each group is expected to contribute a new insight beyond what the previous group said. He claims that this tactic pressures students into having something to contribute to discussion and discourages "free-riders" [11]. We take Woolcock's approach to ensuring compliance with the reading assignments several steps further:

- We assign online reading quizzes to be completed prior to class. Most questions are multiple choice with immediate feedback, but there are always six open-ended "discussion questions" at the end. These quizzes are ungraded, but to earn *any* grade in the course, we state that students must make a "good-faith effort" on every quiz.
- In class, we randomly assign six small discussion groups: one group per discussion question.
- We assign each group a "discussion leader" and assign a grade to the discussion leader based on their response.
- We do not repeat the assigned reading in lecture.

Preece et. al. described software requirements engineering through the lens of interaction design. They describe a "user-centric" approach to gathering requirements for building any GUI. They demonstrate the interplay between human psychology, elicitation techniques, prototyping techniques, and evolutionary design [14]. We assign their book as required reading before each class session and introduce Woolcock's peer-pressure techniques to promote compliance with the reading [11]. Our intent is to frame elicitation and GUI construction as a principled design field, not an ad-hoc field. For example, when students build their GUI prototypes, we require that students build multiple prototypes and describe the tradeoff's they make between Preece's dimensions of human perception and learning. Students later test the alternatives in our school's usability lab. They use the data to empirically decide which prototype to accept. As with Leffingwell's interview template, we emphasize to students that requirements engineering follows well-defined processes that are not ad-hoc.

Macaulay and Mylopoulos reflected on the ideal RE curriculum [9]. They surveyed the pedagogical practice of the time and compared it to expectations of practitioners. They suggested five themes for a good RE learning experience: i) RE is not an isolated activity, but needed at all stages of software development, ii) Requirements can and will change; design software so it can be changed, iii) Requirements come from multiple sources, iv) All aspects of software must be justifiable and traceable, and v) Requirements can and will conflict. We made these themes central to the student experience.

Gabrysiak et al. offer an experience report on using actors to serve as stakeholders for a requirements engineering term project [6]. They recruited graduate students with non-software expertise, introducing a "semantic gap" between the RE students and the actor. Gabrysiak et al name three key goals of a good RE learning experience that informed our work: G_1: "the students should experience a semantic gap during elicitation;" G_2: "the students should experience consistency issues when synthesizing information gathered during an interview;" G_3: "the students should experience the usual problems when validating requirements specified in a way that does not support a suitable view for stakeholders."

Gabrysiak et al's conceptual model of elicitation informed our understanding of how to grade student work when there are actors. In a given elicitation, they expect students to learn some set of facts about the project. The actor's preparation (ability to memorize the facts) caps the set of facts that can be actually recovered. On the other hand, the actor's familiarity with the problem domain (i.e. from their major or internship) allows them to answer questions that they didn't formally prepare for [6]. Based on this model, we graded project work with an emphasis on the quality of the elicitation and analysis, deemphasizing the completeness of the recovered facts.

Callele et al. offer an experience report on introducing students to the consequences of failing to apply RE [2]. They gave 2^{nd} year students typical programming assignments modified to include contradictory and ambiguous requirements. Students initially felt betrayed that the assignments were not "set in stone," but by the end of the term, expressed appreciation for RE. We drew on their work to construct similar assignments.

Delatorre et al. explained the advantages of using client role-playing to generate realism in learning RE [4]. Research on negotiating requirements goes back to Barry Boehm's work in the 1990's [1]. Research recommending specific requirements processes continues to compete with the Agile

movement's emphasis on process reduction, in dictating "what should be taught" in an RE course. For instance, Röder identifies a pattern-based approach for specifying usability requirements [17].

The complexity of requirements elicitation involves not only negotiation skills but also creativity. Maiden et al. identified this need based in clients' inability to understand what they want "till they see it" [10]. So, teaching RE to students includes teaching them to help customers visualize requirements.

Significant research and industry practices are seen in different vocational areas. Palomares et al. provide a catalog of requirements for content management systems [13], Shaban-Nejad et al. propose how to manage requirements changes in healthcare apps [19] and Chanin [3] discuss teaching RE in the context of "startup" projects. These body of works emphasize the varying nature of RE based on project contexts.

III. Teaching Approaches

In this section, we will discuss the evolution of the course and different teaching approaches taken in teaching the course:

A. Internal Clients Similar Projects Model

This is the approach shown for 2003-2006 in Table I. Our intent was to make a realistic experience in gathering and managing requirements, in a required, 10-week course for junior year Computer Science and Software Engineering majors. Reports from industry have shown that software engineers in their first jobs often make assumptions that result in serious project rework. Given a choice, future employers would prefer for students to learn hard lessons about this in school.

The notion of doing something because "a client wants it that way" conflicts with the college ethos that the best ideas spring from your own head. We knew the course would be a hard sell to rationalist CS majors. Toward this end of converting students to empiricists, at the 2003 course inception, we used the following tactics: 1) Team teaching, having two instructors in the room at all times. 2) Each class was a mixture of discussion and application. 3) Team projects with outside clients. 4) Support from visiting industry people, and from case studies demonstrating the value of requirements. 5) Self-validation, via verification testing and prototyping. 6) Client-validation, via feedback in meetings and student presentations. 7) Multiple representations, capturing requirements via problem statements, use cases, supplementary specs etc. 8) Competing interpretations, all teams worked on the same project goal, via different stakeholders, yielding variations due to client preferences.

To us, the clients, and the industry observers, this class was a crucial launch of students into the real world of software development. To our students, the class turned their self-contained world upside down, and they didn't necessarily like it. Teaching it became difficult for instructors, in that they were asking students to grow in unexpected ways, and the loss of autonomy was not appreciated.

Over the next three years, a succession of professors with guidance from our industry sponsors tried to convey the value of the requirements experience to students in this course. The projects moved toward agile, and we began to introduce interaction design and user studies to maximize their exposure to the outside forces which drive software projects. However, one year students responded by reporting their learning as 2.75 out of 5, on the course evaluations. Aside from their rejection of the outside-in knowledge direction, other factors reported by students included the fact that they did not do significant coding in the course, and they felt that their project artifacts, a set of requirements, were incomplete. For our CS courses, accomplishment of a running software project synthesizing the topics is the student measure of having learned something.

Pros: The approach emulated a real-world software engineering project, and it prepared students for a similar but less guided capstone course.

Cons: It was abrupt – The course felt unlike our CS fundamentals courses and called on different skills. We found R = .21 with the data structures course prerequisite, but R = .43 with the technical communications course (N = 359 and 419, respectively). It broke students' long-standing expectation that their CS instructor had the knowledge or deductive powers to solve all the problems presented in the class.

B. External Clients Distinct Projects Model

This was our model for 2007-2011, as noted in Table I. In 2007, our department asked a new, young professor to teach this class. He valued requirements and had a background in interaction design. Students also found him more relatable, which provided an opportunity to strengthen and toughen the course, making our students focus on getting their project work right as a course objective. Previous offerings of the course had focused more on the learning experience and less on ensuring the final product was "right'," or rather, what the client needed and wanted. A tiered learning model was also introduced [12], which enabled students to achieve process goals in their complex projects:

1) Tier 1 – In-class elicitation via mock projects and personas.
2) Tier 2 – Elicitation in homework using simple scenarios.
3) Tier 3 – Elicitation through a team-based design project.

The Junior Design Projects, also using outside clients, mixed learning the material for the first time, in this course, with actually delivering the results to those clients, in a later course. The scaffolding of the three tiers helped overcome the problem of students learning new material and applying it simultaneously to a real project.

After teaching from this model for several years, we felt the requirements course had achieved its main mission for our industry stakeholders. We also believed we had overcome student resistance to learning about requirements – not by watering down that message, but by making it more difficult, while providing paths to success and lots of encouragement.

During this era we tweaked the model to see what additional software development realities could be rolled-in. In fall of 2008-2009, we tied the requirements gathering strongly to project management (PM), teaching a required course on this subject at the same time as our requirements course, and using the PM course to develop plans for the team projects common to both courses. That endeavor was short lived partly because

of a curriculum change – to making a course other than PM be required for CS-only majors in the department.

Pros: The choice of professor and scaffolding of material eased students into the rocky world of eliciting requirements. Having the project started in this class and continue through another provided students with the expected feel that the work will go to completion. Over these years, the good requirements developed by students in this course specifically on that subject did lead to successful deliveries of projects, an activity integrated into follow-up courses in the junior year.

Cons The projects selected for teams of 3-4 students could not reliably produce similar levels of difficulty in requirements gathering. Even more, they could not be guaranteed to stage problems which provided challenges in design and construction, which were taught in the follow-up classes. Another major drawback for this project style was the use of outside clients. As their projects, needs, and understanding of the software process could not be controlled, students did not achieve consistent learning experiences across projects.

C. Single Client Single Project Model

This is the approach shown for 2011-2014 in Table I; its first year overlapped with the prior model. The tier-focused learning in that approach provided reinforcement of important RE concepts throughout the course for students. Nevertheless, feedback from alumni and from the graduating seniors indicated that a successful learning experience to a certain extent depended on the quality of the course project, the complexity of the project, and the amount of time the client was willing to spend with the team. The success we had enjoyed, with the previous approach, emboldened us to push the boundaries even further. We wanted our students to have a more meaningful and realistic project experience, one that required collaboration among multiple development teams. Requirements engineering is part art and part science and teams can learn from each other. In order to handle previous students' feedback and introduce an aspect of cooperative learning, we introduced significant changes to the course project and materials as follows:

Nature of the Project: A typical project in the past was smaller in scope, had features that were mostly canned to help with understanding RE concepts, and had no functional products as deliverables. Even with external clients, the scope of projects were limited to gathering requirements and creating user-interface prototypes. We continued the model of collaboration with colleagues who taught the Software Design course in the Winter quarter and Software Maintenance in the Spring quarter to carry the project forward throughout a full academic year. With the popularity of agile software development methodologies, the waterfall-like requirements phase in the requirements course felt incomplete. Therefore, we introduced the following changes to the course:

1) The overhead of running several projects and manging several clients was a major concern, on top of developing and teaching the course materials for the instructors of these courses. Instead of having multiple clients, one for each project group, we used one client per section.
2) A typical class at Rose-Hulman has on average 25 students in it. A key challenge in selecting projects was that each project must have enough features to make it a meaningful learning experience for all 25 students. We solicited project proposals from our alumni and industry partners. We selected projects that had enough features for the entire class as well as required students to learn and use modern software frameworks. The key idea was to treat the class as a software development company that worked for a client with different sub-teams handling different feature sets.

Requirements Elicitation: Requirements were gathered through client meetings. Client meetings happened on Fridays during class time. Each week, teams took turns to lead the client meeting sessions. This gave a chance for each team to observe how other teams were eliciting requirements and learn from each other. Irrespective of which team was leading the meeting a given week, all teams had a chance to contribute to the agenda and questions to be addressed with the client during the meeting. Each team had to prepare a list of agenda items/questions/descriptions and email it to the coordinating team by 5:30 PM on Wednesday of that week. The coordinating team prepared the final agenda and made it available to the instructor and the client by 3:30 PM on Thursday of that week. The coordinating team conducted the meeting and gave time to the rest of the teams to handle any unanswered client questions. If there were more questions, those were posted to a private GoogleGroups forum, which served as a message board for the class. The client and/or teams answered those questions by the end of the weekend on GoogleGroups.

Product Development Process: We wanted to adopt an agile software development methodology for the course project. In particular, we chose Scrum as it had facilities to manage development cycles of multiple teams using Scrum of Scrums. The details of Scrum can be found in the Essential Scrum textbook that we adopted for the course [18]. In what follows, we present our adoption of Scrum for the course project.

After students self-selected themselves into teams, we formed a Scrum of Scrums (SoS) team, which was composed of a representative from each team. The job of the SoS team was to oversee product backlog grooming (periodic updating of product backlog and effort estimation for user stories using story points unit [18]) and to keep teams up-to-date with the needs and demands of the client. The SoS representatives, thus, served the Product Owner role in their corresponding team advocating for the client. Each team also had a Scrum Master who oversaw daily scrums. Those were short progress update meetings; if a team required an engineering discussion or more detailed discussion, they scheduled a separate meeting to do so with only the members who were relevant to the topic.

A sprint ran for two weeks. At the end of each sprint, the class did a retrospective meeting where they reflected on what worked and what did not in terms of both product and process. The SoS team then recommended changes, which got adopted after reaching consensus among all teams. A sample product backlog taken from a class section is shown in Figure 1.a. The figure also shows the story point estimation for each user story,

Fig. 1: Scrum methodology applied to the project.

as well as the total points for the sprint. Figure 1.b shows the details of a user story. Each user story specified "conditions of satisfaction" (CoS) [18] for each affected team. As a team member made progress on the user story, she also checked off the CoS entry for her team. This way, we were able to consolidate and monitor the overall progress of a particular user story for all teams. Furthermore, the user stories were often backed up by mockups, wireframes, and other relevant document/diagrams to capture the client's intent.

Pros: The students liked the overall real-life project experience and self-learning that happened through the project. They were realistically dealing with complexity of scale. The course model simulated the feel of a real software company. They also had to collaborate and coordinate development cycles among teams. For instance, all front-end teams (Android, iOS, and Web) depended on the backend team to complete their tasks, and the quality assurance (QA) team depended on front-end teams to complete their tasks. There was a lot of learning about requirements but also about the larger context of managing a software product involving several development teams. Furthermore, students gained realistic design and coding experience in this class and were not limited to RE only experiences.

Cons: There are several challenges to teaching this model of the course. First, the varying nature of projects in the multiple sections made it hard to tailor homework assignments to a specific class to help with the project work. Also, certain projects were inherently harder than others and the instructors had to manage the expectation of students, clients, and themselves when things did not get done in time. To make this model of course better, we recommend establishing three deadlines: one for the backend team, another for front-end teams, and yet another for the QA team. However, the logistics of managing such schedules with the overhead of creating and teaching course materials can be overwhelming. Not all teams can be graded using the same rubrics in this setting. To make matters worse, students got better at playing the blame game - teams blaming other teams for not getting their features done. Sometimes they did have a real issue, while other times, they were not proactive in reaching out to other teams to find out about the progress of the dependent features. This was a strong indicator of their lack of motivation to work on someone else's idea without getting paid for the work they put in. Furthermore, the teaching load on the instructor was substantial, as he/she had to work as the front-line project manager, coordinating meetings with clients, coordinating student groups, and facilitating development.

Grading students' work had a challenge of its own in this model. There were a lot of unanswered questions. How would you judge students who got blocked by the other project teams and could not make progress towards their features? When students make a mistake, how do we handle it? Is it a teachable moment from which students learn, or should there be a stronger penalty than a reduced grade, perhaps something that would be practiced in industry. With the realistic nature of the course also came realistic problems of administering it.

D. Entrepreneurship-Based Model

In an experiment, we tried a much-different approach for 2015, as noted in Table I. The previous section explained that the large projects, while realistically simulating industry, tended to suffer because of insufficient student motivation.

Evaluations revealed that student's did not feel excited about working on somebody else's idea. Students also pointed out that the coordination between several teams imposed too much process even in the presence of agile methodologies.

We decided to adopt a process that had minimum rituals and was flexible enough to work under uncertain and changing circumstances, yet simple enough to be understood by all stakeholders of the project. Our experiences with RE also indicated that students would learn better if they can experiment with ideas, receive validity on their ideas quickly, and iterate to refine their projects. These together suggested a process that had three parts:

1) Learn fast through customer interviews, customer development, root cause analysis, and customer acceptance.
2) Build fast using open source components, incremental deployment, continuous integration, unit testing, etc.
3) Measure fast using split-tests, usability tests, and traffic monitoring.

The process aligned closely with the core principles of the Lean Startup methodology [16]. Thus, we adopted lean methodology in the course and required students to propose their own startup idea. Each student would pitch their idea to the class, and recruit classmates to work on the idea for the rest of the quarter. Students were taught to use various tools and techniques to assist in the evaluation of prototypes, performing feasibility analysis, and managing risks with the stated goal of providing incremental value to the project's stakeholders. Each student proposal was vetted for viability and approved by the instructor. Students then performed an initial feasibility analysis of their ideas including: i) determining existing needs ii) assessing strengths and weaknesses of existing approaches, and iii) identification of stakeholders and the benefit provided by the project to said stakeholders.

The feasibility analysis documents were reviewed by the faculty and student project managers. The accepted proposals were then made available to all the students. Students with accepted proposals were then allowed to recruit their colleagues to work in their project using an approach similar to the popular TV show "Shark Tank" by offering ownership stakes in the startup. Student feedback on these proposals were gathered using surveys on the class website.

The rest of the term was divided into several milestones. Students were required to develop a Minimum Viable Product (MVP) at the end of each milestone as follows:

- *Milestone 1:* The main purpose of this milestone was to start documenting requirements for the startup. Student teams were required to interact with identified stakeholders through direct observations and interviews. The results of the interactions were captured using an affinity diagram and user stories. They were further required to identify epic-level user stories and use them to drive the feature-level user stories. For each feature-level user story, the student teams captured storyboard diagrams as the first MVP.
- *Milestone 2:* The main purpose of this milestone was to introduce students to prototyping and evaluation. Student teams were instructed to create two separate low fidelity designs of their system. Each of these designs were to be evaluated by four different users. Feedbacks from the users were captured and findings and changes to the prototypes were documented. Based on the feedback, students coalesced the designs into one final low-fidelity prototype as the second MVP.
- *Milestone 3:* The main purpose of this milestone was to create high-fidelity prototypes building on the low-fidelity prototypes. The teams had to create user interface screens using the actual programming languages that were targeted by the system. We also required teams to start developing the backend and to plan rest of the project. All software development was done using a continuous integration pipeline that teams setup using an in-house Git server.
- *Milestone 4, 5:* In Milestones 4 and 5, teams continued coding the product. Each milestone functioned as a week long sprint. They used standard software engineering practices such as automated testing and continuous integration (CI) and delivery (CD).
- *Milestone 6:* In this milestone teams conducted a usability study of the MVP developed in the previous milestone. The final deliverable for this milestone was a Usability Report that included the following:
  1) Process (Informed Consent, Pre- and Post-Test Questionnaires, Tasks and Goals, and User Demographics)
  2) Findings with evidence in the form of video observation
  3) Recommendations
  4) Analysis (Quantitative analysis of task completion times and other counters from usability reporting software, Qualitative analysis of pre and post-test questionnaire)
  5) Revised High Fidelity Prototype based on usability study. Each screen included a summary of changes made.

Pros: This approach solved a key problem of the past approaches - "the lack of student motivation". Students consistently pointed out that they cared about the project and were invested to work harder. The use of lean methodology also struck a positive note with them. They pointed out that the process created a realistic environment that aligned closely with a startup venture, which made the course interesting.

Cons: The use of CI and CD created several problems that got in the way of learning RE. Teams spent several hours trying to setup their CI/CD pipelines that delayed the observation of user's interaction with products and thereby the refinement of requirements. Also, the project selection process required that several project ideas had to be pruned, which meant that some students had to work on somebody else's idea. This reduced the overall motivation of some students.

The learning associated with requirements happened because of the process that was used and wasn't direct. While the end goal of teaching students about requirements engineering was achieved, it does lead to the question - is there a better model that directly achieves the goal? Furthermore, while students noted that the lean startup process created a realistic environment for a startup and students felt they learned about RE, it's important to note that many of our students do not

go into lean startups. A large portion of our students end up in regulated fields such as aerospace, medical technology, or at large, established companies. The dynamics between stakeholders and developers in these environments is much different than what is seen in a lean startup. In this model, students interviewed the stakeholders they identified, but held most of the power to make decisions about the final product. While students found this version the most motivating, this decision process likely does not reflect the work they will do in the industry. Learning that choices are often not solely in the developer's hands is an important lesson that can be missed in this version of the course.

E. Intentional Learning Model

This approach was taken in 2016, and continues as our current model, as shown in Table I. Learning from previous experiences, we asked: What learning experiences do we want to expose students to during their term project? We synthesized the related work into five desired learning experiences: i) Work on a project in an unfamiliar problem domain [6]. ii) Elicit requirements from a non-CS stakeholder [6]. iii) Experience changing understanding of existing requirements [9]. iv) Experience and resolve conflicts between stakeholders [6]. v) Experience the consequences of failing to apply RE [2].

The experience of "writing code" did not make our list. The related work never claimed that students need to make the connection between code and requirements **through a huge project** - only that the connection needs to be made. We assigned a few early motivational coding assignments to get student buy-in, after which, students stopped coding entirely.

Project setup: Unimpeded by code as the end-deliverable, we created a term project intentionally focused on the above five learning experiences. Our process was: 1) Find a problem domain that few students know about, like medicine. 2) Draw on our industry experience and find a model industry project within the desired problem domain, like blood testing. 3) Build an "oracle" set of final business requirements. 4) Identify stakeholders typical to the problem domain. 5) Distribute the business requirements among these stakeholders. 6) Intentionally introduce overlap and conflict between stakeholders' understanding of the requirements. 7) From the perspective of each stakeholder, plan responses in advance to Leffingwell's interview questions. 8) Organize these responses into scripts/FAQs. 9) Find actors (industry volunteers, drama students, and TAs) and give them the scripts/FAQs.

For example, this year's project involved discovering the business requirements for a software upgrade to a blood testing lab. We identified seven stakeholders:

- Assay Development: This group is constantly doing research to determine new ways to test blood samples.
- Stability: The company produces test kits that are provided to doctors offices for day-to-day tests such as strep and mono. The Stability group's goal is to measure the Stability of these kits over time under different storage conditions. They must keep detailed records under specific conditions to ensure proper results over time in the hopes of detecting possible issues with kits before they fail in customer use.
- Production Testing: The Production Testing group does the day-to-day sample testing of blood samples from patients. They must follow strict procedures and documented processes. The software system must provide checkpoints and information along the way that keeps the user on track, while not bogging down their job with unnecessary steps. This group faces strict FDA regulation.
- Patients: The labs would like to offer test result to patients online. This data must be secured so that only the patient and the doctor who ordered the test(s) have access to it.
- Doctors: The doctors who order tests should be able to retrieve the results of their patients as soon as they are available. Preferably, this can be done online, but some doctors may still prefer a PDF that can be printed or faxed.
- Lab Managers: Lab Managers unlike others would just like to see efficiency of their personnel.
- Analysis Personnel: As new labs are being created, or labs are being run over time, analysis personnel will periodically check the data to look for trends.

Each actor had their own view of the data and their own corner of the overall business process. For example, we provide a portion of the scripts we gave to two different stakeholders regarding sample states in lab testing:

Sample States: Lab Technician If they ask you about sample states (something the production lead likely mentioned):

- Samples start as ordered, which means that the doctor has requested a sample be taken.
- Then, once a phlebotomist has drawn the sample, it moves to collected. If the sample needs to be shipped (so it wasnt collected here) then it stays collected until it gets to our lab, then it moves to the arrived state.
- If the sample was collected here, it moves to arrived once its passed to the lab. Once its been organized into an assay, its noted as scheduled.
- Once the assay has been started it moves into the "In Test" phase, then once we record results it becomes "Tested."
- Once the report is compiled and sent / made available to the doctor, its moved to "Reported."
- Eventually, samples will be disposed of an then they get moved to "Disposed."

Sample States: Assay Developer
If they ask you about the states a sample goes through (this is something that the production lead may have mentioned): Tracking this isn't all that important to it, and if we have to go through the same sample states as production wed probably lose our minds.

Notice the lab technician placed significant emphasis on the business process of moving samples through the different states. Compare that emphasis with this contradictory excerpt from the technician's colleague, an assay developer. Each pair of roles had similar contradictions in scope and priority. We gave the actors these scripts in printed and tablet form, then set them loose on the students. We found the use of tablets to be much easier, as the script was divided into sections and the

actor could click to the desired section based on the question asked, making the flow more like a regular business meeting.

Student learning experience: Unlike previous years, students learned about RE through a flipped format, where they answered preclass quizzes from their reading of Interaction Design [14]. RE can be a dry subject to teach through lecture, so the intent of the assigned reading was to allow students to absorb the material at their own pace. Students came to lab to discuss the quiz's open-ended "discussion questions." They ended each lab with an exercise. The early lab assignments based on [2] were designed to motivate the need for RE. Later lab exercises explored the principles of interaction design as applied to high-fidelity prototyping.

Outside of lab, students worked in teams of four on their term project. Similar to past iterations, we have several milestones that led students from elicitation of the business requirements, into prototyping the UI, and ultimately having students build a release schedule reflecting the stakeholders' priorities. Each milestone had an accompanying rubric mandating that students apply the best-practices from the reading.

Students were given little information about this project upfront. This includes domain information, as it is common to enter a project in industry with no knowledge of the client's domain. The students' objective was to interview the actors to determine the true nature of the project. Students transcribed their interviews and explicitly traced the requirements they discovered back to the raw text of the transcript. As students encountered the seeded conflicts and unintentional ambiguity, they used their RE artifacts as a means to an end: they sent clarification emails to the client's "official" business email, explaining the situation and asking the clients for feedback on their RE artifacts. These email accounts were monitored and responded to by the course professors. We attempted to maintain consistent responses based on the scripts provided to the actors. Scripts were expanded if a student asked a question that exposed missing information in the script.

We used the clarification emails to expound on the requirements and correct actors' errors, but most importantly, to introduce conflict in stakeholders' priorities. For example, one team established that it was a low priority for lab technicians to describe the expected values of positive blood tests. When the assay developer was asked to review the students' priorities, we gave this response:

> Team,
> Adding expected results to assays is of paramount importance to the science. I can't believe you would list it last. How is the humble production lab technician supposed to know whether they did the right thing if we don't specify at least the expected value and acceptable bounds? We have a higher calling that starts with informing the production lab technician what to expect.

Additionally, responding to client emails opened up an opportunity for us to see how students managed written communication with a client. Some students would blame the client for miscommunication by quoting what was said in a previous meeting, referring to lines in transcripts, etc. While this may prove that the client did contradict themselves, we were able to introduce an important point to our students: Clients often do not understand what developers need to know, as developers do not understand what the client does all day. We were also able to provide coaching on better ways to communicate in an email.

Pros: The course's subtle pressures to make students read the textbook succeeded. Students expressed that motivational assignments helped illustrate the need for RE. Removal of coding portion of the project gave students time to see the necessity of eliciting requirements to navigate conflict and ambiguity. Conflicts between stakeholders' needs forced students to realize that different people have different needs. This realization manifested in the students' UI prototypes, which gravitated to different views for different users.

Cons: We had no way to evaluate students' elicitations, only their deliverables. Our students felt tricked by the motivational assignments. We would like to reduce the "trick-question" feel of the initial assignments. The project was more instructor-intensive than the entrepreneurial model because the instructors had to set up the actors and respond to client email. The entrepreneurial model scaled better in this respect. The actors were often overwhelmed by the novelty of the problem domain, particularly during their first few elicitations. Our actors had their scripts in tablet form, yet navigating them during the interview was tedious, though not as much as paper. Students felt that further practice and organizational assistance would help the actors convey their material. Transcribing the numerous elicitation interviews was a major pain point. We insisted on students transcribing the raw audio to help provide traceability. Students would benefit from a transcribing service.

IV. Evaluation

To open this discussion, we point out that one must be bold enough to try different flavors of a course, which might have different advantages and disadvantages, in order to have alternatives whose results one can compare. One also must persist in including assessment material, so that each progressive change can serve as a baseline for the next.

We conducted two forms of assessment of the various versions of our requirements engineering course. The first form of assessment was conducted during the delivery of the course. Each course version included an anonymous in-class assessment. We conducted a survey at the 3.5 week mark and 7 week mark of a 10 week class. The survey included the following questions:

- What do you like about the class?
- What would you like to see change about the class?
- What do you like about the way this class is taught?
- What would you like to see change about the way this class is taught?

Students were instructed to limit the answers to these questions to the day-to-day delivery of the class and any changes that could be made mid-stream. As such, feedback from these immediate impact surveys are not reported here. Students were instructed to maintain a journal of their feedback and to

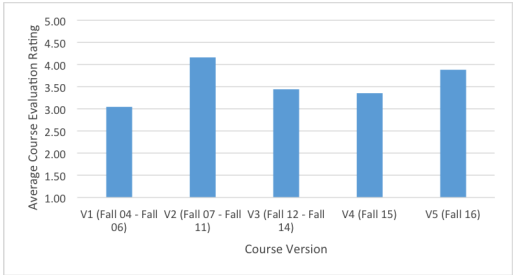

Fig. 2: CSSE 371 Overall Learning by Version

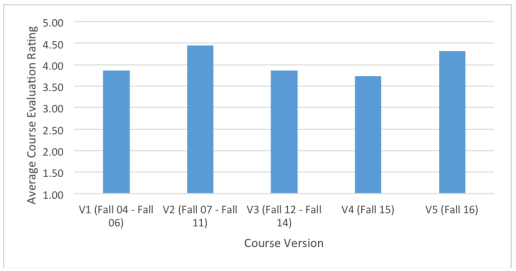

Fig. 3: CSSE 371 Course Evaluation Comparison of Lab Material and Teaching Tools by Version

recommend changes/suggestions in a summative assessment that was carried out at the end of the class.

The end of term summative assessment is comprised of the following questions:

1) Please rate the quality of your learning in the class
2) The laboratory assignments and the course material reinforced one another
3) The work load for this course in relation to other courses was
4) Overall, how would you rate this class
5) The professor was well prepared for this class
6) The professor used teaching techniques that helped me learn
7) The professor was available for help outside of class
8) The professor was genuinely interested in teaching the class
9) Please rate the professor's overall performance in the class.

The number of students completing the course evaluations for the various versions is as follows:

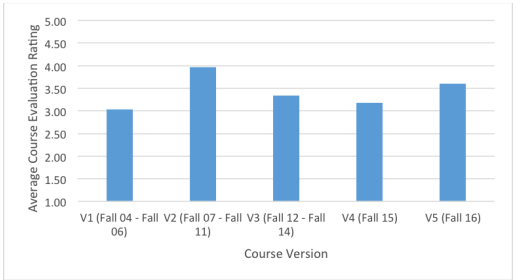

Fig. 4: CSSE 371 Course Evaluation Comparison of Overall Course Rating by Version

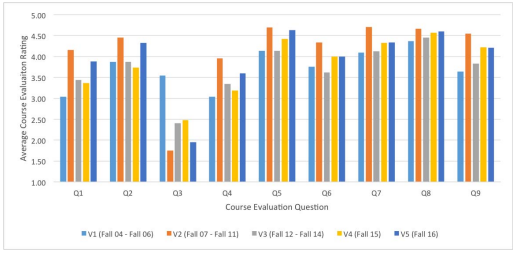

Fig. 5: CSSE 371 Course Evaluation by Version

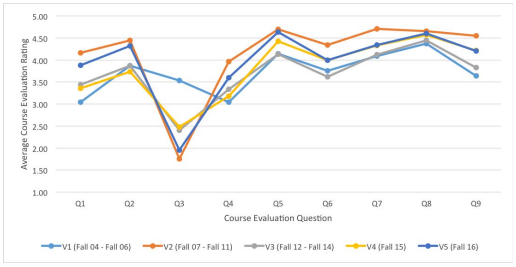

Fig. 6: CSSE 371 Course Evaluation by Version

1) V1 - Internal Clients Model (III-A) - 100 Students
2) V2 - External Clients Model (III-B) - 148 Students
3) V3 - Single Client Single Project Model (III-C) - 170 Students
4) V3 - Entrepreneurship Model (III-D) - 92 Students
5) V4 - Intentional Learning Model (III-E) - 98 Students

Q1, Q2, Q4, Q5, Q6, Q7, Q8, Q9 were evaluated using a 5 point Likert scale with the following options: a) strongly agree - 5, b) agree - 4, C) neither agree nor disagree - 3 , d) disagree - 2, and e) strongly disagree - 1.

The results from the student survey are provided to the instructor in the form of a mean and standard deviation. Figures 2, 3, and 4 provide the means obtained from Q1, Q2, and Q4, respectively, for the various versions of the class. The earliest version of the class (internal clients) III-A had the weakest performance for overall learning and had the weakest overall course rating. The poor learning quality led us to the development of the external clients model described in III-B. This version of the class had the best performance across the board. Some of the higher performance can be directly attributed to the instructor as noted by the better performance of the instructor in questions 6, 7, 8, 9 as shown in Figures 5 and 6. The problems with the external clients model directly led to the development of the next two versions of the class. While learning and overall course rating dipped a little, the class and the topics in question were still rated highly by the students. The latest version of the class with an emphasis on intentional learning has helped raise student's perception of learning and overall class ratings.

Q3 is a measure of workload and is evaluated using a 5 point Likert scale with the following options: a) much lighter - 5 , b) lighter - 4, c) about the same - 3, d) heavier - 2 and e) much heavier - 1. Lower the rating, heavier the workload in

that version of the class and as shown in Figures 5 and 6, we have been able to consistently maintain a comparable workload (rated by the students as heavier than the average course at the institution) across the various versions of the class.

In addition to these questions, students also had the option to provide free form responses to the following questions:
1) Explain why your learning was at this level
2) Describe one or more strengths of this course
3) Describe one or more ways this course can be improved
4) Explain why you gave the instructor this rating.

The responses to these free form questions have been captured in the Pro's and Con's discussions in the previous subsections. Response from the students and alumni on free-form questions is filled with positive feedback about the quality of learning, material and the course in general. A similar sentiment has been observed by the senior capstone instructors. They have noted the increased competency of the students in eliciting and managing requirements and change. More details are not included due to space constraints, but we will be happy to share the results upon request.

V. Threats to Validity

One of the threats to validity is the viability of student journaling, the process we used as a setup for student evaluations at the end of the course. In theory, recording at the time is much preferred to relying on recollections. However, in our world today, any blogging-type activity also puts students into the same "set" in which they do social media commentary. This cannot be ignored as a cause of what we see in our results. Even if we provide them with guidance and training, we are competing with an influence which they may interact with 100 times a day.

In using Student Evaluations of Teaching (SET's) as an assessment tool, we also assume that students can self-report their degree or quality of learning. For convenience, we often pretend that all teaching situations can be leveled, so as to compare these reports. But, in fact, people unfamiliar with subjects cannot be expected to provide expert reports, and the more unfamiliar they are, the farther off the reports are likely to be, probably in ways that do not average-out. For most of the computer science students taking a course in requirements, this is the first time they have been challenged to think in a Gestalt way, to see things from other people's perspectives in order to succeed, to deal with ambiguous assignments, and to question their own judgment. We cannot rule out the possibility that students learned more in this class than they think they learned, and that they will become aware of this gain only when they take on serious projects in the future. We see this effect in the same students' senior projects.

VI. Conclusion

At this point in the evolution of the course, we are tempted to conclude that realistic, problem-based learning approaches for a first course in requirements, present difficult situations for both students and instructors. We have opted, instead, for a scripted course for which the challenges can be still strong but better controlled.

We are pleased with the recent adoption of a flipped classroom in this course. Here, the format motivates students to study material on their own, and it makes maximum use of class time to do problem-solving.

Providing a sufficiently convoluted project, which emphasizes the right requirements lessons, remains dicey. The problem is that a "canned" project is already familiar to the instructors and some instructors may not choose to portray the searching thought processes, which lies at the heart of requirements analysis. On the other hand, inventing a new problem for each class, and implementing all of its aspects, like role-play situations, is a large amount of work. These issues were listed in the "cons" for our current course model.

We invite others to consider any or all the flavors of this course that we have tried. As noted, each had unique pros and cons. Variables as fundamental as the nature of your CS students, or the amount of prep time you have available, could make one of these alternatives preferable.

References

[1] B. Boehm and A. Egyed, "Software requirements negotiation: Some lessons learned," in *ICSE*, 1998, pp. 503–506.

[2] D. Callele and D. Makaroff, "Teaching requirements engineering to an unsuspecting audience," *SIGCSE Bull.*, vol. 38, no. 1, pp. 433–437, 2006.

[3] R. Chanin, L. Pompermaier, K. Fraga, A. Sales, and R. Prikladnicki, "Applying customer development for software requirements in a startup development program," in *SoftStart*, 2017, pp. 2–5.

[4] P. Delatorre and A. Salguero, "Training to capture software requirements by role playing," in *TEEM*, 2016, pp. 811–818.

[5] H. Femmer, D. M. Fernández, E. Juergens, M. Klose, I. Zimmer, and J. Zimmer, "Rapid requirements checks with requirements smells: Two case studies," in *RCoSE*, 2014, pp. 10–19.

[6] G. Gabrysiak, H. Giese, A. Seibel, and S. Neumann, "Teaching requirements engineering with virtual stakeholders without software engineering knowledge," in *REET*, 2010, pp. 36–45.

[7] R. L. Glass, *Software Runaways: Monumental Software Disasters*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998.

[8] D. Leffingwell and D. Widrig, *Managing software requirements: a use case approach*. Addison-Wesley, 2003.

[9] L. Macaulay and J. Mylopoulos, "Requirements engineering: An educational dilemma," vol. 2, no. 4, pp. 343–351.

[10] N. Maiden, S. Robertson, and J. Robertson, "Creative requirements: Invention and its role in requirements engineering," in *ICSE*, 2006.

[11] Michael J. V. Woolcock, "Constructing a syllabus." [Online]. Available: https://tinyurl.com/hvx7qbv

[12] S. Mohan and S. Chenoweth, "Teaching requirements engineering to undergraduate students," in *SIGCSE*, 2011, pp. 141–146.

[13] C. Palomares, C. Quer, X. Franch, S. Renault, and C. Guerlain, "A catalogue of functional software requirement patterns for the domain of content management systems," in *SAC*, 2013, pp. 1260–1265.

[14] J. Preece, Y. Rogers, and H. Sharp, *Interaction design: beyond human-computer interaction*, fourth edition ed. Wiley.

[15] G. Rempel, "Defining standards for web page performance in business applications," in *ICPE*, 2015, pp. 245–252.

[16] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[17] H. Röder, "A pattern approach to specifying usability features in use cases," in *PEICS*, 2011, pp. 12–15.

[18] K. S. Rubin, *Essential Scrum: A Practical Guide to the Most Popular Agile Process*, 1st ed. Addison-Wesley Professional, 2012.

[19] A. Shaban-Nejad and V. Haarslev, "Towards a framework for requirement change management in healthcare software applications," in *OOPSLA*, 2007, pp. 807–808.

[20] Y. Yu and H. Sharp, "Analysing requirements in a case study of pairing," in *AREW*, 2011, pp. 4:1–4:6.