The Journal of Systems & Software 144 (2018) 366–381

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

On a pursuit for perfecting an undergraduate requirements engineering course

Chandan R. Rupakheti*, Mark Hays, Sriram Mohan, Stephen Chenoweth, Amanda Stouder

Rose-Hulman Institute of Technology, 5500 Wabash Avenue, CM 100, Terre Haute, IN 47803, United States

ARTICLE INFO

Keywords:
Requirements engineering
Project-Based learning
Course evolution

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 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, several project-based approaches have been experimented with in teaching RE, and these have evolved over time. In this paper, the progress of teaching methodologies is documented to capture the pros and cons of these varied approaches, and to reflect on what worked and what did not in teaching RE to undergraduate engineering students.

1. Introduction

It is well-known that engineering of requirements is not a simple task. To prepare students for a successful software engineering career, requirements engineering is strongly emphasized in the Rose-Hulman Institute of Technology curriculum. This paper describes the development of a junior year course in Requirements Engineering, a course which is mandatory 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. Most in the field agree that incoming students do better if they recognize college as an extension of their learning in high school. Similarly, most CS programs offer a capstone course in the senior year to ease students' entry into professional life after graduation. Engineering schools try to maximize job fit at graduation. While there are 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 CS and SE majors go straight into software development work, so preparation for this career is the 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 the CS and SE majors are experiential, and students believe they "know something" when they can do it. Thus, when the Requirements Engineering course was invented, students had to do work which simulated real world settings. This course has been updated, from 2003 onward, to reflect both the experiences from teaching the subject and the research support for courses of this type. For example, research support provided the emphasis placed on 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 (Macaulay and Mylopoulos, 1995). In Section 2, current research in requirements education is summarized.

Table 1 highlights the stages of the course's development. During different eras, the course 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

* Corresponding author.
E-mail addresses: rupakhet@rose-hulman.edu (C.R. Rupakheti), hays@rose-hulman.edu (M. Hays), mohan@rose-hulman.edu (S. Mohan), chenowet@rose-hulman.edu (S. Chenoweth), stouder@rose-hulman.edu (A. Stouder).

https://doi.org/10.1016/j.jss.2018.07.008
Received 21 December 2017; Received in revised form 22 May 2018; Accepted 3 July 2018
Available online 04 July 2018
0164-1212/

Table 1
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

example, in 2015 students were allowed 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 the "Integrated" team formation had entire classes of 20 or more working on the same project, divided into sub-teams.

Section 3 describes the thought that went into each of these varying approaches. The section describes the basic pedagogical mechanisms and the pros and cons of the outcomes. Section 4 describes the evolution of assessment in the course. Section 5 evaluates data collected at the time the classes were taught. Section 6 revisits the validity of the results, and Section 7 concludes the paper.

2. Related work

This section describes studies backing the current format of the authors' RE course ("the RE course"), studies in support of earlier versions of the course, and studies showing approaches of others in teaching RE.

2.1. The trickiness of RE activities in the real world

Eliciting and managing requirements is a well-known wicked problem, with new needs and problems becoming apparent as the work progresses. Saiedian and Dale report on the existing methods and obstacles in Requirements Elicitation (Saiedian and Dale, 2000). There are a variety of knotty concerns which could be studied in the classroom to help bring home the general underlying open-endedness of RE:

Release scheduling: Svanberg, et al. study the issue of deciding which requirements go into which release (Svahnberg et al., 2008). The authors suggest that students may be able to play the role of industrial subjects, in their classes.

Fault context: Mendonca, et al. describe how system faults related to functional and non-functional requirements should be understood in terms of their context, to know the seriousness of the fault (Mendonça et al., 2014). This applies in particular to dependability, an issue not well addressed in undergraduate classes anyway. E.g., At this level of education, student-designed and other systems generally are not studied over time or built to be durable.

Global RE: Berenbach explains that organizational and other management issues overshadow technical requirements problems, and in distributed organizations dealing with the former issues these are primary to process improvement (Berenbach, 2006). Berenbach asserts that "soft topics" like organizational considerations are more important – this goes beyond the sensed charter of most CS departments.

Access: Nakatani and Tsumaki say the maturity (completeness) of requirements depends on the accessibility of the source of requirements to the person tasked with eliciting these (Nakatani and Tsumaki, 2011). This is a messy problem, because there are multiple stakeholders in a project, not all equally close to the people charged with capturing requirements. This complexity perhaps must be experienced to be understood. It is likely difficult to simulate in the classroom.

Reevaluation: Silva Souza, et al. point out the compromises often necessary in the "real world" – Operationalizing requirements includes actions like relaxing them when they are violated too regularly (Souza et al., 2012). It is hard to imagine undergraduate students, in general, being prepared to understand this scenario.

The need for tackling RE in school is part of a larger industry issue of providing software engineering skills. Radermacher, et al. ask managers and hiring personnel where recent CS graduates struggled – among the areas were communication skills with co-workers and with customers, and a lack of project experience (Radermacher et al., 2014). Section 2.5 "RE pedagogy basics," below, characterizes the issues of RE in greater detail.

2.2. Creativity in eliciting requirements

The complexity of requirements elicitation involves not only negotiation skills but also ingenuity. Maiden et al. identifies this need based in clients' inability to understand what they want "till they see it" (Maiden et al., 2006). So, teaching RE to students includes teaching them to help customers visualize requirements. Dalpiaz et al. say that requirements should be tied to root-level goals of an organization, after which they can be represented as an operationalization of those goals (Dalpiaz et al., 2014). This clearly requires deep knowledge of the goals and expression of requirements in a domain-specific way. Shaw et al. claim that software designers need to get "out of the box" and understand good solutions in real-world ways (Shaw et al., 2005). This includes understanding what clients need and also constraints imposed by the client's context.

2.3. Difficulties in teaching RE

Authors comment continually about the problem of getting across the value of RE to students lacking substantial software work seasoning. Callele, et al. offer an experience report on introducing students to the consequences of failing to apply RE (Callele and Makaroff, 2006). They gave second-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, they expressed appreciation for RE.

Memon et al. (2010) confirmed in 2010 that ways to address these issues of preparation for RE were still "in progress." What was the best practical approach to bridge the experience divide? Challenges to teaching SE topics include lack of students' intra-team communication skills, student immaturity and lack of time management skills. Waiting for undergraduates' brains to mature would be a last-ditch remedy.

2.4. Course methodology - in general

Woolcock describes 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" (Woolcock, 2006). The authors of this paper take Woolcock's approach to ensuring compliance with the reading assignments several steps further:

- *Pre-class work due:* Online reading quizzes are assigned 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 RE course, students must make a "good-faith effort" on every quiz.
- *In-class application:* In class, six small discussion groups are

randomly assigned: one group per discussion question.
- *Leadership experience:* Each group is assigned a "discussion leader" and a grade is given to the discussion leader based on their response.
- *Individual responsibility:* The assigned reading material is not repeated in lecture.

2.5. RE pedagogy basics

Macaulay and Mylopoulos reflected on the ideal RE curriculum (Macaulay and Mylopoulos, 1995). 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:

1. RE is not an isolated activity.
2. Requirements can and will change; thus one should design software for change.
3. Requirements come from multiple sources.
4. All aspects of software must be justifiable and traceable.
5. Requirements can and will conflict.

The authors of this paper believe an RE course should either deal with these five basic issues of practice, or explain why one of them must be excluded.

2.6. Course methodology - Case studies

Case studies are a classic approach to addressing complex, realistic situations, in law schools and business schools as well as engineering schools. Many resources exist for case studies emphasizing requirements issues (Femmer et al., 2014; Glass, 1998b; Leffingwell and Widrig, 2003; Rempel, 2015). The authors of this paper use case studies throughout the SE curriculum and in the RE course specifically. Section 3.2.4 "Case Study Discussions," below, notes several relevant case studies. Advances in engineering typically are made by studying failures, and this is straightforward when it is not one's own failure. Having students share in understanding of the limits of practice frames other learning activity in a course.

Teaching requirements elicitation: Leffingwell and Widrig's book presents a framework for eliciting software requirements. They provided a template titled "Context-Free Interview" for constructing a 1-1 interview with any stakeholder. The RE course uses this Context-Free Interview to frame elicitation as an algorithm, not an ad-hoc conversation (Leffingwell and Widrig, 2003). Leffingwell and Widrig describe multiple means of requirements elicitation. However, the authors of this paper believe that students in a beginning class should practice only one of these thoroughly.

2.7. Course methodology - Interaction design (ID)

There are specific ID recommendations in the literature, in terms of teaching RE. For example, Schneidewind, et al. suggest a persona technique to give better understanding of user needs (Schneidewind et al., 2012). The use of personas and accompanying "epics" is now a common tool to focus feature development in agile software groups.

Preece et al.'s book portrays software requirements engineering through the lens of interaction design. The authors 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 (Preece et al., 2006). The RE course assigns topics from their book as required reading before each class session and introduces Woolcock's peer-pressure techniques to promote compliance with the reading (Woolcock, 2006). The intent is to frame elicitation and GUI construction as a principled design field, not a capricious field. For example, when students build their GUI prototypes, the RE course requires that they build multiple prototypes, and that they describe the tradeoffs they make between Preece's dimensions of human perception and learning. Students later test the alternatives in the school's usability lab. They use the data to decide empirically which prototype to accept.

2.8. Course methodology - Agile:

If RE is a part of a software engineering course, then the overall development process typically is introduced first. If, however, RE is a first course taken in software engineering, then some process needs to be used as a framework for understanding where RE fits, and so as to organize course project work. For Agile methods, there are many resources available, including online ones. The RE course used Rubin's book (Rubin, 2012) as a reference.

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. Inayat et al. (2015) say the benefits of agile are interesting to explore in comparison to the needs of RE: Being agile about requirements solicitation has become very popular, but studies assessing the strengths and weaknesses of this approach have not provided much guidance.

Agile methods have had a tendency to ignore non-functional requirements such as availability and security – requirements whose resolutions require a sweeping view to make the remedies effective. The traceability of requirements over time also can suffer, because of the episodic nature of iterative development. The intentionally reduced emphasis on documentation does not help this problem.

Hvatum and Wirfs-Brock (Hvatum and Wirfs-Brock, 2015) analyze in depth the problem of agile's "magic backlog," and they offer solutions: especially ripping the assumption that a typical project backlog represents "a well-founded set of detailed requirements." Teams may believe these user stories and such only need to be broken down into manageable chunks. An informal process for gathering these becomes even more problematic with large, complex system development.

The challenge of representing the entire product in the backlog includes showing where unknown functionality likely resides. Temporary items of this type can then be exchanged for detailed items when they are elaborated.

Hvatum and Wirfs-Brock say the backlog usually does not represent what features are expected for what release of a product, and the simple act of ignoring as a "YAGNI" [You aren't going to need it] or not accounting for what work may be coming down the line incurs technical debt. The authors recommend an Application Lifecycle Modeling (ALM) tool for showing all the connections to different required types of work, system design, and development cycles. Shared queries and reports add to visibility for all interested parties.

The authors say a further communication problem lies in the disappearance of "done" features from the backlog once they are implemented and tested. This act lessens visibility for what has to be regression-tested. An additional lack of organization comes from the listing of requirements in the backlog in order of customer development priority, which tends to reduce the understanding of relationships among requirements. The authors recommend, instead, showing the backlog as a model of the product you are building, especially depicting how each requirement contributes to larger pieces of functionality.

Getting this full view of agile "RE issues to be concerned about" goes beyond what a single undergraduate course could likely teach. Indeed, some of these issues go beyond what one could picture as solvable incrementally via periodic team retrospectives, even for a seasoned agile team. Adding-in process structure that should have been there from the start – that sounds like a project redo.

The above discussion of agile RE demonstrates that the problem of doing this well is still unresolved. The goals of a starting class on the subject will need to be modest, particularly if one includes active learning to support all the topics covered.

2.9. RE pedagogy variations

Here are some points of difference in teaching approaches that have been taken, or could be taken, as described in the literature:

2.9.1. Learning a context well:

Scepanovic and Beus-Dukic (2015) present an RE teaching curriculum emphasizing requirements discovery, where students learn about the application domain and stakeholders' perspectives. As stated in Section 2.9.14 "Teaching RE in varying contexts," below, many authors feel that learning a particular domain is as valuable as being technically adept at eliciting requirements generally. By learning one domain well, students can experience this perspective and perhaps transfer the lessons to additional domains.

2.9.2. Teaching the reality of soft skills:

Portugal, et al. say that obstacles in teaching RE are related to the nature of RE – it is multi-disciplinary and part of both CS and social sciences (Quintanilla Portugal et al., 2016). The RE course includes the latter part, with real client interaction. Sedelmaier and Landes add that RE needs to be taught in an authentic way, mirroring real-world complexities. It involves non-technical skills and practice with real customers is best (Sedelmaier and Landes, 2017). The RE course utilizes ID resources to bring in elements of psychology and sociology.

2.9.3. Using RE patterns:

Röder identifies a pattern-based approach for specifying usability requirements (Röder, 2011). da Silva et al. (2015) define a pattern language for use case creation, and emphasize the practicality and value of use cases for all stakeholders and members of the development group. Their patterns are built on Withall (2007) requirements patterns and on da Cruz (2014) earlier use case patterns. These authors all had varying reasons for creating patterns, such as textual specification versus modeling. Emphasis on patterns would provide continuity with this popular way to approach the teaching of object-oriented software design.

2.9.4. Frameworks and checklists:

Affleck and Krishna (2012) say that the adequate elicitation, analysis, and management of non-functional requirements depends on having a dependable framework to guide these activities. In general, such a system should be based on systematically obtained numbers, to provide firm goals for development. Such frameworks exist, or can be created for class. E.g., The lists of important requirements for quality attributes (non-functional requirements) given in the classic software architecture book by Bass et al. (2012).

2.9.5. Bad smells:

Femmer et al. (2014) also note how analysis of requirements issues can be systematized. Finding bad requirements quickly requires detection methods – a list of "bad smells" can be useful as a lightweight analysis tool. E.g., Bad smells such as "ambiguous adverbs and adjectives," or open-ended terms like "provide support." This approach shares part of the familiar conceptual framework of refactoring.

2.9.6. Scaffolding:

Mohan and Chenoweth (2011) described how to scaffold the learning of requirements skills, using progressively harder problems. This approach also is sketched in the current paper. The current incarnation of the RE course abandons that approach in favor of students' applying skills directly to their project. Section 3.2 describes the advantages and disadvantages of scaffolding.

2.9.7. Role-playing:

Delatorre et al. explained the advantages of using client role-playing to generate realism in learning RE (Delatorre and Salguero, 2016). Research on negotiating requirements goes back to Barry Boehm's work in the 1990's (Boehm and Egyed, 1998). Section 2.9.13, provides additional guidance.

2.9.8. Entrepreneurial:

Grbac et al. (2015) point out how abstract reasoning about requirements, as well as architecture, only comes into play in developing and evolving large-scale systems. These authors feel students may be able to propose their own system or their own business idea, as a basis for gathering such more complex requirements. The entrepreneurial approach, which the present authors used for one year, is based on the "lean" company formation ideas by Ries (2011). See Section 3.4 for details.

2.9.9. Ethnography:

Cunningham and Jones (2005) recommend an ethnographic approach but admit this is extremely time consuming. A shortcut is "autoethnography": "In autoethnography or 'personal ethnography', ethnographic techniques of observation and analysis are applied to one's own experiences; the challenge is to view oneself objectively, to see one's own worldview as freshly as possible and to then interpret the identified experiences in the light of applicable theory." (Crawford, 1996)[1] In industry, a standard startup trick for building new kinds of systems is to "live with the customer" while doing the design and development. This ruse is emulated in agile processes by having a customer representative on-site. In student projects, it is difficult to find real customers willing to spend the same amount of time teaching software people their business, because of the unlikely outcomes. The problem of introducing ethnography to an RE course remains open.

2.9.10. The large, coordinated project:

Coppit and Haddox-Schatz (2005) describe how an entire class worked on a single software engineering project, to add realism. Short deadlines were used to prompt coordination of work completion among sub-teams. Because of project size, delicate aspects of the project were not known until students were working on them – they had a real "wicked problem." Communications among sub-teams and requirements traceability were issues. The project management role for the instructor was intense. This mirrors the present authors' experience with the approach, described in Section 3.3.

2.9.11. Emphasizing special requirements:

Ludi (2007) said the need for contextual requirements to be made clear is exemplified by the question of whether or not software engineering students consider accessibility requirements in the interaction design of their system. A higher level of compliance will, understandably, be achieved if stakeholders with disabilities are included in the RE process for a student project.

2.9.12. A systems engineering approach:

Chang et al. (2010) describe how a SysML-based solution for documenting requirements is needed to clarify the requirements in a compound system, such as software built for multicore embedded systems. Process customization based on the nature of the complexities can be desirable. An RE course taught in this way would bring-in systems engineering learnings.

2.9.13. Using actors as stakeholders:

Gabrysiak et al. offer an experience report on using actors to serve as stakeholders for a requirements engineering term project (Gabrysiak et al., 2010). They recruited graduate students with non-software expertise,

[1] The authors agreed that autoethnography should be used to supplement interacting with other stakeholders, and that it should be done when students really are familiar with a domain.

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 affect our work:

> G1: "the students should experience a semantic gap during elicitation;"
> G2: "the students should experience consistency issues when synthesizing information gathered during an interview;"
> G3: "the students should experience the usual problems when validating requirements specified in a way that does not support a suitable view for stakeholders."

Sections 3.2 and 3.5 describe experiences with using actors.

Gabrysiak et al.'s conceptual model of elicitation inspired a means 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 Gabrysiak et al. (2010). The instructors of the RE course graded project work with an emphasis on the quality of the elicitation and analysis, deemphasizing the completeness of the recovered facts.

2.9.14. Teaching RE in varying contexts

Palomares et al. provide a catalog of requirements for content management systems (Palomares et al., 2013), Shaban-Nejad et al. propose how to manage requirements changes in healthcare apps (Shaban-Nejad and Haarslev, 2007) and (Chanin et al., 2017) discusses teaching RE in the context of "startup" projects. This body of works emphasizes the varying nature of RE based on project contexts.

Proynova et al. (2011) say stakeholder requirements for health care information systems are influenced by personal and social factors such as personal values. Song et al. (2006) note that there is confusion in the software industry supporting healthcare, regarding the workflows needed for decision making. There need to be tools depicting this workflow in a way which all parties can understand.

Gonzales et al. (2009) discuss using another area as the basis, one typically associated with service learning projects. In developing systems for non-profit service work, such as caregivers of children with autism, expressions of support for this community are crucial to gaining meaningful requirements.

3. Teaching approaches

This section discusses the evolution of the course and different teaching approaches. Table 2 summarizes the pros and cons of the approaches.

3.1. Internal clients similar projects model

This is the approach shown for 2003–2006 in Tables 1 and 2. The course exposed students to a realistic experience in gathering and managing requirements. The course took shape as a required, 10-week course for junior year Computer Science and Software Engineering majors. Software engineers in their first jobs often make assumptions that result in serious project rework. Given a choice, prospective employers prefer that students learn hard lessons about faulty assumptions while still in school.

College students are taught to innovate to find new solutions to problems, but Requirements Engineering teaches that the client has the last word in what gets implemented. The many restrictions of the business world, such as time, funding, government regulation, legacy systems, etc., place unexpected restrictions on applications that students have yet to encounter. The notion of doing something because "a client wants it that way" was a hard sell to rationalist CS majors. Toward this end of converting students to empiricists, at the 2003 course inception, the following tactics were used: (1) Team teaching, having two instructors in the room at all times. The two instructors played contrasting roles, providing different perspectives on covered concepts to the students. (2) Each class was a mixture of discussion and application. (3) Team projects with outside clients. (4) Support from visitors in the industry, 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, but via different stakeholders, yielding variations due to client preferences.

To the instructors, the clients, and the industry observers, this class launched students into the real world of software development. To students, the class turned their self-contained world upside down, and they didn't necessarily like it. Student resistance to growth made teaching the course difficult for instructors.

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 began to introduce interaction design and user studies to maximize 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

Table 2
Overview of approaches and associated pros and cons.

Pros	Cons
Internal Clients Similar Projects Model (2003–2006)	
- Emulated a real-world setting	- The course felt abrupt, unlike other CS fundamentals courses
- Prepared students for capstone experience	
External Clients Distinct Projects Model (2007–2011)	
- Continuity of project in follow up courses	- Could not produce similar levels of difficulty in RE as before
- Better scaffolding	- Inconsistent learning experience due to project variations
Single Client Single Project Model (2011–2014)	
- Very close to real world	- Teams blocked other teams
- Students learned to manage the complexity of multiple teams	- Grading students' work was challenging
- Agile requirements with design and coding activities	- Heavy load on instructor to manage projects as well as teaching
Entrepreneurship-Based Model (2015)	
- Implemented startup ideas by adopting the Lean methodology	- Some Lean subprocesses interfered with learning RE
- Students care more about the projects because they were their own ideas	- Not all students' ideas were realized
Intentional Learning Model (2016 - Present)	
- Succeeded in making students read the textbook	- Actors were often overwhelmed by the novelty of the problem domain
- Highly focused on RE by removing coding activities	- Evaluation of students' elicitations is still a concern
- Successfully introduced conflicts between stakeholders for students' learning	- Transcribing the numerous elicitation interviews was a major pain point

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 Rose-Hulman CS courses, accomplishment of a running software project synthesizing the topics is the student measure of having learned something.

3.1.1. Pros

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

3.1.2. Cons

It was abrupt – The course felt unlike our CS fundamentals courses and called on different skills. We tested the strength of the relationship between final grades in our requirements course, and final grades in its two prerequisites, the data structures course and the technical communications course. We found R = 0.21 (correlation coefficient) with the data structures course (population size = 359), but R = 0.43 with the technical communications course (populations size = 419), both results significant at the 0.05 level of confidence.

3.2. External clients distinct projects model

This was the model for 2007–2011, as noted in Tables 1 and 2. In 2007, the 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 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 (Mohan and Chenoweth, 2011), 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.

3.2.1. Tier 1 - In-class elicitation

Gathering requirements is an art as much as practice. The course aimed to provide students with several opportunities understand, try out, and master their elicitation skills, by utilizing the notion of personas and mock projects to facilitate this in the first tier of the learning model.

- Mock Projects: Class discussions were organized around a mock project scenario with the instructor and teaching assistants playing the roles of various stakeholders. The scenarios placed the students in different situation, and helped drive the relevance and applicability of various elicitation techniques. The students were provided with a brief vision of the product and then had to work their way through needs, features, use cases and prototypes during the various class sessions. Here is a sample problem statement that was used:
  A system that integrates micro-blogging services like Twitter to run in-class assessments and to gather feedback during class. This could be generalized into a system that integrates with PowerPoint and helps gather student feedback in classes. Faculty can conduct quizzes, gather data via polls, and conduct plus / delta evaluations.
- Personas: Students were provided with a description of these personas and trained faculty played the roles of people with these personas, and students were then required to work in groups to elicit requirements of a very small project.

3.2.2. Tier 2 - Elicitation via homework projects

The in-class elicitations were mostly done using team oriented active learning exercises. To provide students with more opportunities to practice and reflect, as well as to serve up a framework for self assessment, homework assignments were used as the second learning model. The homework was introduced with the following scenario:

Bank de Fleur was established in 1875 to assist employees of Fleur Polytechnic to manage their money and investments. It has now grown to be one of the most successful banks in the Wabash Valley. ***Bank de Fleur*** *has ATMs for convenient access at various locations on campus and in several locations downtown and in the adjoining rural areas. They have recently released plans for redesigning and updating their ATM software to enable a better user experience for their customers and to enhance the overall security.*

Teaching assistants were enlisted to represent the bank's management and IT divisions. Some students participating in the senior capstone experience were used to play the role of the Bank's customers. Each actor was provided with a list of features and they were instructed to answer only the questions posed by the students.

The assignments were designed to help the students gain experience in the following areas:

1. Problem and stakeholder identification
2. Feature identification
3. Storyboards and Low-fidelity prototypes
4. Use cases to capture software requirements
5. Non-functional requirements
6. Performing usability tests and re-design based on user feedback

3.2.3. Tier 3 - elicitation through junior design

Project proposals are gathered throughout the school year from alumni, companies that visit Rose-Hulman career fairs and local not-for-profit organizations. It should be noted that these organizations **do not** pay a fee to participate in the junior design sequence and are appraised of the educational objectives associated with the junior design sequence.

The project proposals are made available to the students and the final teams are formed based on student input and interest. The team is retained for the entire sequence of classes that comprise the junior design sequence. To help the student teams as they worked on their projects in the RE course, the work was divided into five distinct milestones, as follows:

1. Milestone 1: Current system analysis, client stakeholder analysis, feature listing, and problem statement
2. Milestone 2: Use cases, data flow diagrams, and domain models
3. Milestone 3: Supplementary specifications: non-functional requirements and initial UI design
4. Milestone 4: Change control plan, coding standards, and acceptance test cases and plan
5. Milestone 5: Usability test plan, test report, and revised UI design

Teams were required to turn in a draft to their project manager (TA) a week prior to the final submission.

3.2.4. Case study discussions

In parallel to the core three-tier course structure, case studies were utilized to add a flavor of real-world lessons to the class. Students were assigned case studies to read ahead of the class discussion. The activity was divided into two parts - (a) a pre-discussion report due before the start of class, and (b) an in-class discussion using the concepts covered in class with an emphasis on identifying reasons behind the success/failure of a project. The list of case studies include the following:

1. BAE Automated systems: Denver Airport baggage handling system (Montealegre et al., 1996)
2. Why (some) large computer projects fail ? - An analysis of the FAA Advanced Automation System (Glass, 1998a)
3. When professional standards are lax: The Confirm failure and its

lessons (Oz, 1994)
4. Anatomy of the Boston Big Dig project
5. Requirements engineering challenges in multi-site software development organizations (Damian and Zowghi, 2003)

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

During this era the model was tweaked to see what additional software development realities could be rolled-in. In fall of 2008–2009, requirements gathering was strongly tied into the project management course (PM). This was done by teaching a required course on PM at the same time as the requirements course, and using the PM course to develop plans for the team projects common to both courses. That endeavor was short lived partially because of a curriculum change – to making a course other than PM be required for CS-only majors in the department.

3.2.5. 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.

3.2.6. 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.

3.3. Single client single project model

This is the approach shown for 2011–2014 in Tables 1 and 2; its first year overlapped with the prior model. The tier-focused learning in that approach reinforced important RE concepts. 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 previously enjoyed emboldened instructors to push the boundaries even further. Students should 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. The next sub-sections describe how the instructors responded to this feedback.

3.3.1. 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 in the RE course was limited to gathering requirements and creating user-interface prototypes. Collaboration was done with colleagues who taught the Software Design course in the Winter quarter and who taught 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, the following changes were introduced to the course:

1. The instructors framed the course as a software development/consulting company.
2. The instructors became project managers. Each instructor managed a different project.
3. The instructors solicited project proposals from alumni and industry partners and selected only sufficiently large projects. A typical classroom at Rose-Hulman has on average 25 students in it. Students needed a project with enough features to engage in a meaningful learning experience.

Early on in the quarter, when the teams worked on identifying requirements, the teams collaborated in the requirements gathering process. The approach also made the teams feel like a real world company where multiple development teams were working on different aspects of the same product.

In particular, almost all selected projects required an implementation in Web, Android, and iOS platforms. Students self-organized into the following development teams: Web, Android, iOS, Backend, and Quality Assurance. Such student teams are also known as *component teams*. The other way to break down teams was based on features. The pros and cons of these two approaches are discussed at the end of this section.

3.3.2. Requirements elicitation

Students gathered requirements through client meetings. Client meetings happened on Fridays during class time. Each week, teams took turns to lead the client meeting sessions. This rotation gave each team a chance to observe how other teams were eliciting requirements and to 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 to questions to be discussed 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.

3.3.3. Product development process

With the popularity of agile software development, the instructors adopted a Scrum methodology, 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, which the instructors adopted for the course (Rubin, 2012). This section presents the adoption of Scrum for the course project.

After students self-selected themselves into teams, a Scrum of Scrums (SoS) team was formed, 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 the story points unit Rubin, 2012) and to keep teams up-to-date with the needs and demands of the client. Thus, the SoS representatives 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.

Each sprint ran for two weeks. Within a sprint, students worked with items taken from the product backlog. A sample product backlog

Fig. 1. Scrum methodology applied to the project.

from the students is shown in Fig. 1a. The figure also shows the story point estimation for each user story, as well as the total points for the sprint. Fig. 1b shows the details of a user story. Each user story specified "Conditions of Satisfaction" (CoS) (Rubin, 2012) 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 workflow consolidated students' status updates into a central location.

Students applied a rapid prototyping development cycle to develop their features. First, students built prototypes (mocks, story boards, and wireframes) to gauge the client's expectations and get rapid feedback. Second, students developed the features in their team's platform (Android, iOS, backend, ...). Finally, students demoed the functional product to the client, receiving more feedback. Students repeated this last step until the client expressed satisfaction with the software. In most cases, students completed features after one round of prototyping and two rounds of demos.

At the end of each sprint, the students held a retrospective meeting. Students 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. Each student rated the effort of all team members, including himself. These ratings, the progress status reflected by the Trello board, and commits on Github assessed the overall performance of a team member in the project.

3.3.4. 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 course and were not limited to RE only experiences.

3.3.5. Cons

The instructors encountered several challenges while teaching this model of the course. First, grading students' work was challenging in this model. Certain projects were inherently harder than others, so sometimes "upstream" teams blocked other teams from doing their assigned work. How do you assign a grade to a blocked team? When students make a mistake that stalls the project, how do you grade that? 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? One could mitigate this problem of blocked teams by forming feature-based teams rather than component-based teams. A feature-based team could implement a vertical slice of the system independently of other teams. Each team member could self-select whether they work on the frontend, backend, or automated testing work for the team. In most cases, the instructors opted not to use feature teams because the instructors wanted team members all working together to learn the same new technology, like Android or iOS. Consequently, different component teams needed different grading rubrics.

Second, the instructors had to manage the expectation of students, clients, and themselves when students failed to complete work on time. As teams became blocked, students got better at playing the blame game - they blamed the other teams for not getting their work done. Sometimes, genuinely challenging technical difficulties blocked teams. Other times, students failed to communicate their needs to the other teams. The instructors inferred that students lacked motivation to work on someone else's idea without getting paid for the work they put in.

Finally, the teaching load on the instructor was substantial. The instructor were the front-line project managers, who not only co-ordinated meetings with clients and student groups, but also facilitated development. The logistics of managing such schedules with the overhead of creating and teaching course materials was overwhelming.

3.4. Entrepreneurship-Based model

In an experiment, a much-different approach was tried in 2015, as noted in Tables 1 and 2. The previous section briefly mentioned that the

large projects, while realistically simulating industry, may have suffered because of insufficient student motivation. Evaluations revealed that students 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. Additionally, the large project approach had made the students' junior project inherently more complex than the upcoming senior project.

3.4.1. Nature of the project

The instructors adopted a new 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. Instructor experiences with RE also indicated that students would learn better if they could experiment with ideas, validate 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 (Ries, 2011). Thus, lean methodology was adopted 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 faculty and student project managers reviewed the proposals. Students with accepted proposals then recruited their colleagues to work on their project using an approach similar to the popular TV show "Shark Tank" by offering ownership stakes in the startup.

3.4.2. Product development process

The term was comprised of several milestones. Students developed a Minimum Viable Product (MVP) at the end of each milestone as follows:

- *Milestone 1:* Student teams interacted with identified stakeholders through direct observations and interviews. They documented them using affinity diagrams and user stories. Students also identified epic-level user stories and used those to drive the feature-level user stories. The student teams captured feature-level storyboard diagrams as the first MVP.
- *Milestone 2:* Students began prototyping and evaluation of their idea. Students created two separate low fidelity designs of their system and asked 4 different users to evaluate their designs. Based on the feedback that the users gave, students coalesced the designs into one final low-fidelity prototype as the second MVP.
- *Milestone 3:* Students created high-fidelity prototypes building on the low-fidelity prototypes. The teams created user interface screens using the actual programming languages that were targeted by the system. The teams began developing the backend and planned the rest of the project. Students set up their own Git server and their own continuous integration (CI) and continuous delivery (CD) pipelines.
- *Milestone 4, 5:* Each milestone functioned as a week-long sprint. Students further expanded their product and used automated testing, CI, and CD.
- *Milestone 6:* 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-test and post-test questionnaires)
  5. Revised High Fidelity Prototype based on usability study. Each screen included a summary of changes made.

3.4.3. 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.

3.4.4. 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 users' 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 was not 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 is 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 and 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 students can miss in this version of the course.

3.5. Intentional learning model

The Intentional Learning approach was used in 2016, and continues as the current model, as shown in Tables 1 and 2. Learning from previous years, it was asked: What learning experiences do instructors want to expose students to during their term project? Related work was synthesized into five desired learning experiences: (i) Work on a project in an unfamiliar problem domain (Gabrysiak et al., 2010). (ii) Elicit requirements from a non-CS stakeholder (Gabrysiak et al., 2010). (iii) Experience changing understanding of existing requirements (Macaulay and Mylopoulos, 1995). (iv) Experience and resolve conflicts between stakeholders (Gabrysiak et al., 2010). (v) Experience the consequences of failing to apply RE (Callele and Makaroff, 2006).

The experience of "writing code" did not make the 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. A few early motivational coding assignments were assigned to get student buy-in, after which, students stopped coding entirely.

3.5.1. Project setup

Unimpeded by code as the end-deliverable, a term project intentionally focused on the above five learning experiences was created. The process was: (1) Find a problem domain that few students know about, like medicine. (2) Draw on 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, the 2016 project involved discovering the business requirements for a software upgrade to a blood testing lab. Seven stakeholders were identified:

- 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 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 results 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 for 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, Figs. 2 and 3 show portions of the scripts that were provided to two different stakeholders regarding sample states in lab testing.

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

> 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 we would probably lose our minds.

Fig. 2. Sample states: assay developer.

> 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 and thus moved to "Disposed."

Fig. 3. Sample states: lab technician.

contradictions in scope and priority. The actors were given these scripts in printed and tablet form, then they were set loose on the students. The use of tablets was 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.

The actors were **not** the instructors. The instructors recruited the Lab Managers from actual software engineers, managers, and engineering management professors who expressed interest in helping students learn RE skills. The instructors recruited the second round of technical stakeholder roles (Assay Development and Production Testing) from Rose-Hulman Drama Club students. Rose-Hulman does not have a theater degree program; the actors were all STEM majors who simply aspired to act a few times.

3.5.2. 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 (Preece et al., 2015). 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 Callele and Makaroff (2006), 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, there were several milestones that led students from elicitation of the business requirements, into prototyping the UI, and ultimately to 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 up-front. 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. Instructors attempted to maintain consistency in email responses by basing content on the scripts provided to the actors. Scripts were expanded if a student asked a question that exposed missing information in the script.

The clarification emails were used 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, the following response was given:

> 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, an important point was introduced to the students: clients often do not understand what developers need to know, as developers do not understand what the client does all day. This opened the door for instructors to provide coaching on better ways to communicate in an email.

3.5.3. 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.

3.5.4. Cons

- There was no way to evaluate students' elicitations, only their deliverables.
- Students felt tricked by the motivational assignments. Reducing the "trick-question" feel of the initial assignments would be good for student morale.
- 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. 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. Students were required to transcribe the raw audio to help provide traceability. Students would benefit from a transcribing service. Current versions of the course allow for the use of these services.

4. Assessment

This section describes how student assessment work evolved over the years. In the early years of the course, rubric-based grading was applied. As the burden of creating relevant rubrics for new types of assignments grew, it moved to a points-based, no-zero grading policy. An attempt was then made to address problems with each method of grading by piloting Nilson's "specifications-based grading" (Nilson and Stranny, 2014) in the most recent version of the course.

4.1. 2003–2011: rubric-based grading

The assignments in these versions of the course emphasized the students' abilities to elicit and analyze requirements from clients. The course was assessed through work done by students on homework, in-class exams, and a final project. The homework and exams were used to assess individual student ability, while the project emphasized teamwork. The homework built on a crafted scenario with previously identified requirements. The exam presented students with raw interview material and asked them to extract the requirements.

The project was divided into milestones and each milestone was associated with a grading rubric/scale. The rubrics assessed items drawn from Leffingwell and Widrig, such as needs, features, constraints, quality attributes, etc. Leffingwell and Widrig (2003) Since each project served a different client, the rubrics did not provide great detail and had variability built into them to compensate for project eccentricities. The instructor also individualized the rubrics as needed based on each project. Each individual milestone contributed equally to the project grade. The full rubrics remain archived on the CSSE371 course web site. (Mohan and Chenoweth, 2011).

Each team was assigned a project manager who, as a part of their duties, evaluated and provided initial feedback to the teams on their milestones. The mangers were coached on providing feedback on these initial milestones. Milestones were graded by the instructor after this round of initial feedback.

Students' individual contributions to their teams were assessed at mid-term and the end of the class by using surveys. The surveys were completed by both the team members as well as the primary teaching assistant in charge of the team. A weighted average was computed with each individual student's evaluation given a weight of 15% and the teaching assistant's given a weight of 40%. Each team/TA had a total of 20 points to distribute across the four students in a team and the weighted average was then used to drive a multiplier on the teams grade (anywhere from 0.7 to 1.1) to determine the individual student grade.

4.2. 2012-2015: point-based grading

The assignments in the 2012–2015 versions of the course emphasized students' abilities to analyze requirements present in their raw data, not their ability to elicit requirements from people. The course assessed students through the group project, homework, and one take-home mid-term. The group project drove assessment in this version of the course. "Homework" in these iterations scaffolded individual contributions to the group project. For example, one homework assigned individual students to create prototypes for their group projects. Milestone 2 then asked students to combine their individual prototypes into a cohesive whole. Most project milestones followed this paired format. The take-home exam presented students with raw interview material and asked them to extract the requirements.

Each assignment split the students' deliverables up on a 100-point

scale. Graded deliverables often drew from scrum methodology, ie user stories, storyboards, etc. The instructors used a simple no-zero marking scheme to assign points to each deliverable:

- Excellent Work: 100 Meaningful Work: 75
- Some Work: 50 No Work: 0

Assessments never called attention to abstract ideas like grammar or professionalism. The individual instructors assessed violations of these cross-cutting concerns ad-hoc. Grades on group work were adjusted using input from end-of-term peer evaluations. This information was gathered from an online survey in which students were asked to rate themselves and their teammates on a 1–5 Likert scale. Students were also asked to justify each rating. Instructors interpreted these ratings on an ad-hoc basis.

4.3. 2016: specification-based grading

Assessment in the 2016 version of the course departed from previous iterations. While assessment still emphasized the usual group project, homework, and exams, instructors sought to address problems faced with point-based grading. In point-based grading, students could effectively skip assignments critical to building understanding, yet earn passing grades that did not truly reflect their level of understanding. To try to resolve this issue, ideas from Nilson's specification-based grading (Nilson and Stranny, 2014) were extended to implement what the course called outcome-based grading. Every assessment in the course was traced to one or more of the course learning outcomes. Student-facing rubrics were built to map outcome-related deliverables to letter grades.

Fig. 4 lists the Project Milestone 1 rubric. The assignment mapped to Learning Outcome 2 in the gradebook. To earn an A on Milestone 1, students had to complete the A-level deliverables as well as the B, C, and D-level deliverables to the "satisfaction of the instructor." When students seemingly completed the A-level deliverables, but perhaps failed to capture a key B-level requirement like "meeting the true need," they earned only a C. Knowing that students would invariably mess up the fundamentals, Nilson's retry system (Nilson and Stranny, 2014) was used to give students a fixed number of retries on assignments. This model was pitched to students as "choose your own grade," because theoretically students who wanted to achieve less could simply focus all their energy on doing the D and C-level deliverables well, while ignoring the B and A level deliverables.

These changes supported an outcome-centric means to compute students' overall course grades. Rather than breaking the overall grades down on homework/project/exam, they were broken down by outcome. Students were required earn a passing grade in every outcome to pass the class. This outcome-based aggregation scheme further emphasized the goal of intentional learning.

There was no systematic effort to distinguish individual student grades on group projects in this iteration of the course. At the end, there was insufficient data to form valid conclusions about individuals' contributions to the group projects.

Outcome 2 (Determine the requirements of stakeholders using standard techniques.)

C:

- Transcript that accurately captures actual interview.
- Business Requirements section.
- Each Business Requirement is explicitly hyperlinked to supporting part of the interview transcript via Google Doc Bookmarks.
- Business Requirements capture essence of basic problem.

B:

- Use Cases section.
- Use Cases follow Preece's [54] basic format:
  - Breakdown of system-user interaction
  - Alternative courses for each step that has potential error cases
- Use Cases and Business Requirements captured all information students might gain through basic lines of questioning.
- Interview fully captured true need of the stakeholder

A:

- Use Cases use best practices from lecture
- Every use case addressed a stated need and every need is addressed by some use case.

Fig. 4. Project M1 rubric.

5. Evaluation

To open this discussion, it is necessary 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 must also persist in including assessment material, so that each progressive change can serve as a baseline for the next.

Two forms of course assessment were conducted, of the various versions of the 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. This survey was conducted 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 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. Q1 - Learning: Please rate the quality of your learning in the class
2. Q2 - Reinforcement: The laboratory assignments and the course material reinforced one another
3. Q3 - Load: The work load for this course in relation to other courses
4. Q4 - Overall: Overall, how would you rate this class
5. Q5 - Preparedness: The professor was well prepared for this class
6. Q6 - Techniques: The professor used teaching techniques that helped me learn
7. Q7 - Availability: The professor was available for help outside of class
8. Q8 - Interested: The professor was genuinely interested in teaching the class
9. Q9 - Performance: 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:

1. V1 - Internal Clients Model (3.1) - 100 Students
2. V2 - External Clients Model (3.2) - 148 Students
3. V3 - Single Client Single Project Model (3.3) - 170 Students
4. V3 - Entrepreneurship Model (3.4) - 92 Students
5. V4 - Intentional Learning Model (3.5) - 98 Students

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. The lower the rating, the heavier the workload in that version of the class and as shown in Figs. 8 and 9, all versions have been able to consistently maintain a comparable workload (rated by the students as heavier than the average course at the institution).

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. Figs. 5–7 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), 3.1, had the weakest performance for overall learning and had the weakest overall course rating. The poor learning quality led to the development of the external clients model described in 3.2. 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 Figs. 8 and 9. 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.

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.

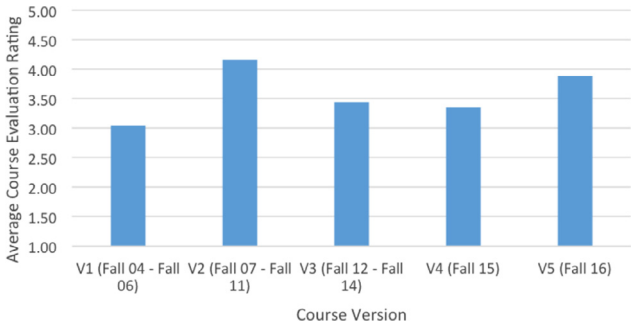

Fig. 5. CSSE 371 overall learning by version.

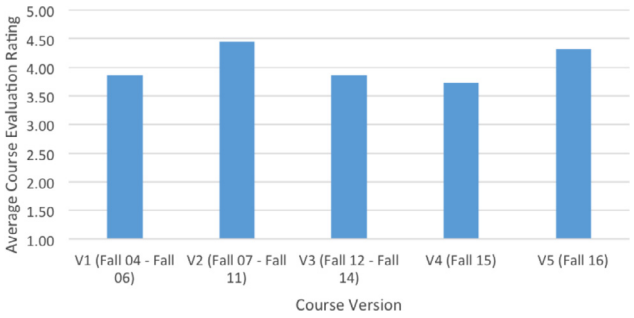

Fig. 6. CSSE 371 course evaluation comparison of lab material and teaching tools by version.

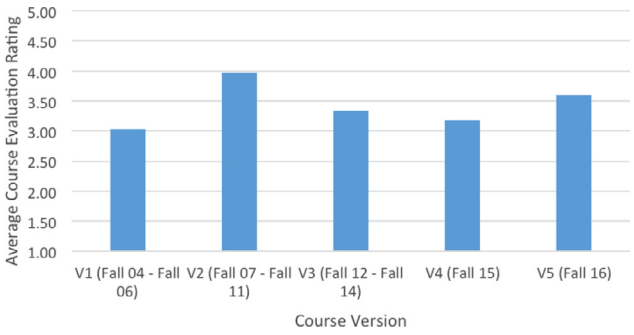

Fig. 7. CSSE 371 course evaluation comparison of overall course rating by version.

6. Threats to validity

One of the threats to validity is the viability of student journaling, the process 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 the 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 is seen in the results. Even if students are provided with guidance and training, the review process is 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, it is assumed that students can self-report their degree or quality of learning. For convenience, it is often pretended 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. The possibility that students learned more in this class and that they will become aware of this gain only when they take on serious projects in the future cannot be ruled out. This effect is seen in the same students' senior projects.

The result aggregates performances of several instructors teaching different sections of the course. The performances of the instructors can vary in such situations. Furthermore, a student's familiarity with a course instructor can make a course evaluation less objective.

7. Conclusion

At this point in the evolution of the course, it is tempting to conclude that realistic, problem-based learning approaches for a first course in requirements, present difficult situations for both students

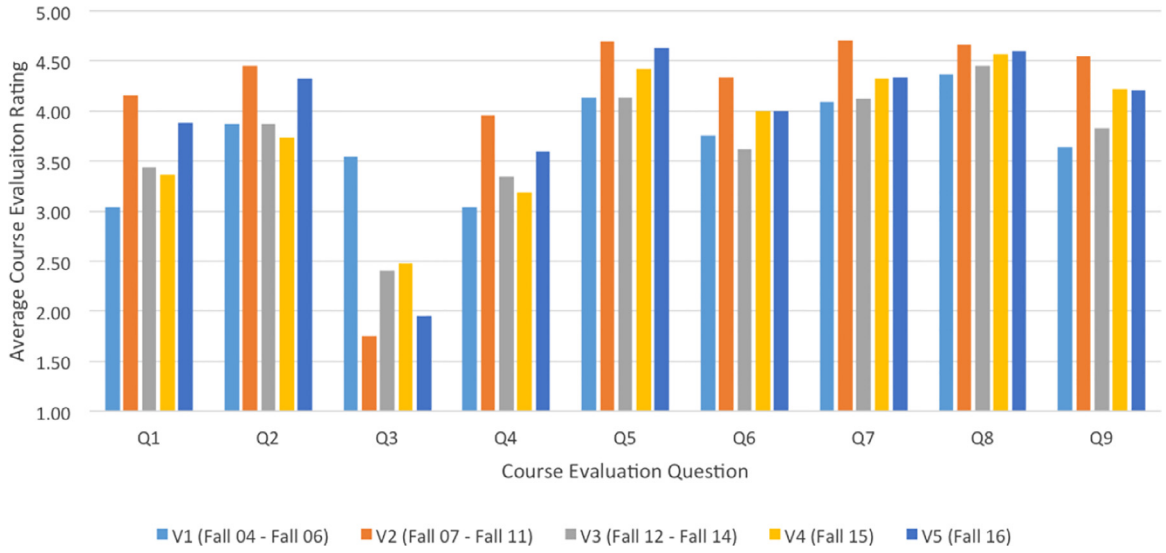

Fig. 8. CSSE 371 Course Evaluation by Version Q1 - Learning, Q2 - Reinforcement, Q3 - Load, Q4 - Overall, Q5 - Preparedness, Q6 - Techniques, Q7 - Availability, Q8 - Interested, Q9 - Performance.

and instructors. Instructors have opted, instead, for a scripted course for which the challenges can be still strong but better controlled.

Instructors 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.

All such courses share the issue that, as successive, different course topics are addressed, these topics will have greater or lesser value for any particular project. By tying projects to an external client, one loses some of the control to inject relevance artificially, because the client may not want this. For example, if a client decides during the software design course that they want their project written in a non-object-oriented language, and satisfying that client is the overriding end goal, then a very large chunk of project applicability can be lost. And, it may not be possible to foresee such issues back in the preceding spring or summer when project selection was done.

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 process, 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 the current course model.

The *External Clients Distinct Projects* Model and the *Single Client Single Project* Model both tried to provide a project that was valuable in three successive courses – the Requirements course that was taught in the fall term, a Software Design course which followed in the winter, and a Software Construction and Evolution course that was taught in the spring. For clients, the key goal was to complete their project over a year, as is done in most software capstone courses, but with the foundational teaching done simultaneously in each of those three courses. This approach shared many of the advantages and disadvantages of a classic, year-long undergraduate "software engineering " course that tries to tie all the learning to a full-year project. The approach also characterizes many problem-based learning programs, where there is an effort to tie learning to a single, large project.

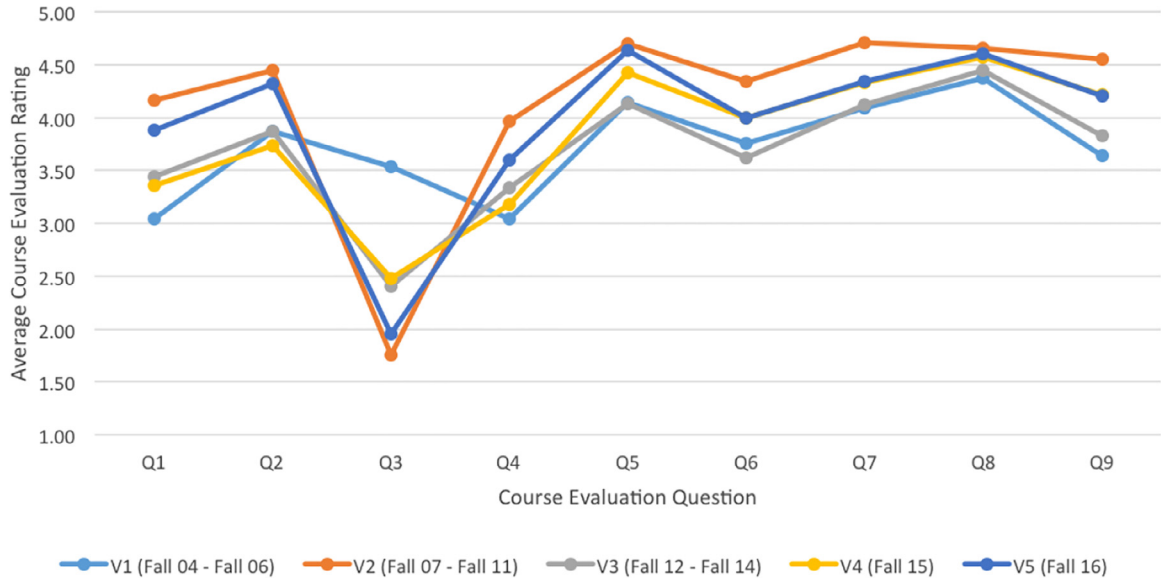

Fig. 9. CSSE 371 Course Evaluation by Version V1 - Internal Clients Model, V2 - External Clients Model, V3 - Single Client Single Project Model, V4 - Intentional Learning Model.

Others are invited to consider any or all the flavors of this course. As noted, each had unique pros and cons. Variables as fundamental as the nature of the specific CS students, or the amount of prep time available, could make one of these alternatives preferable. What works well for one professor may not work equally well for another in a different situation. Understanding the pros and cons of varied approaches to teaching requirements engineering provides instructors with the information necessary to choose the course format that works best in their situation.

References

Affleck, A., Krishna, A., 2012. Supporting quantitative reasoning of non-functional requirements: a process-oriented approach. Proceedings of the ICSSP. pp. 88–92.

Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice, 3rd edition. Addison-Wesley Professional.

Berenbach, B., 2006. Impact of organizational structure on distributed requirements engineering processes: lessons learned. Proceedings of the GSD. pp. 15–19.

Boehm, B., Egyed, A., 1998. Software requirements negotiation: some lessons learned. Proceedings of the ICSE. pp. 503–506.

Callele, D., Makaroff, D., 2006. Teaching requirements engineering to an unsuspecting audience. SIGCSE Bull. 38 (1), 433–437.

Chang, C.-H., Lu, C.-W., Hsueh, N.-L., Chu, W.C., Shih, C.-H., Yang, C.-T., Hsiung, P.-A., Koong, C.-S., 2010. Sysml-based requirement modeling environment for multicore embedded system. Proceedings of the SAC. pp. 2224–2228.

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

Coppit, D., Haddox-Schatz, J.M., 2005. Large team projects in software engineering courses. SIGCSE Bull. 37 (1), 137–141.

Crawford, L., 1996. Personal ethnography. Commun. Monogr. 63 (2), 158–170.

da Cruz, A.M.R., 2014. A pattern language for use case modeling. Proceedings of the MODELSWARD. pp. 408–414.

Cunningham, S.J., Jones, M., 2005. Autoethnography: a tool for practice and education. Proceedings of the CHINZ. pp. 1–8.

Dalpiaz, F., Souza, V.E.S., Mylopoulos, J., 2014. The many faces of operationalization in goal-oriented requirements engineering. Proceedings of the APCCM. pp. 3–7.

Damian, D., Zowghi, D., 2003. Requirements engineering challenges in multi-site software development organizations. Requir. Eng. 8 (3), 149–160.

Delatorre, P., Salguero, A., 2016. Training to capture software requirements by role playing. Proceedings of the TEEM. pp. 811–818.

Femmer, H., Fernández, D.M., Juergens, E., Klose, M., Zimmer, I., Zimmer, J., 2014. Rapid requirements checks with requirements smells: two case studies. Proceedings of the RCoSE. pp. 10–19.

Gabrysiak, G., Giese, H., Seibel, A., Neumann, S., 2010. Teaching requirements engineering with virtual stakeholders without software engineering knowledge. Proceedings of the REET. pp. 36–45.

Glass, R., 1998a. Software Runaways. Prentice Hall.

Glass, R.L., 1998b. How not to prepare for a consulting assignment, and other ugly consultancy truths. Commun. ACM 41 (12), 11–13.

Gonzales, C.H., Leroy, G., de Leo, G., 2009. Requirements engineering using appreciative inquiry for an online community of caregivers of children with autism. Proceedings of the SAC. pp. 142–146.

Grbac, T.G., Car, v., Vuković, M., 2015. Requirements and architecture modeling in software engineering courses. Proceedings of the ECSAW. pp. 36:1–36:8.

Hvatum, L.B., Wirfs-Brock, R., 2015. Patterns to build the magic backlog. Proceedings of the EuroPLoP. pp. 12:1–12:36.

Inayat, I., Moraes, L., Daneva, M., Salim, S.S., 2015. A reflection on agile requirements engineering: solutions brought and challenges posed. Proceedings of the XP. pp. 6:1–6:7.

Leffingwell, D., Widrig, D., 2003. Managing Software Requirements: A Use Case Approach. Addison-Wesley.

Ludi, S., 2007. Introducing accessibility requirements through external stakeholder utilization in an undergraduate requirements engineering course. Proceedings of the ICSE. pp. 736–743.

Macaulay, L., Mylopoulos, J., 1995. Requirements engineering: an educational dilemma. Automat. Softw. Eng. 2 (4), 343–351.

Maiden, N., Robertson, S., Robertson, J., 2006. Creative requirements: invention and its role in requirements engineering. Proceedings of the ICSE.

Memon, R.N., Ahmad, R., Salim, S.S., 2010. Problems in requirements engineering education: a survey. Proceedings of the FIT. pp. 5:1–5:6.

Mendonça, D.F., Ali, R., Rodrigues, G.N., 2014. Modelling and analysing contextual failures for dependability requirements. Proceedings of the SEAMS. pp. 55–64.

Mohan, S., Chenoweth, S., 2011. Teaching requirements engineering to undergraduate students. Proceedings of the SIGCSE. pp. 141–146.

Montealegre, R., Nelson, H., Knoop, C., Applegate, L., 1996. BAE automated systems (a): denver international airport baggage-handling system. Harvard Business School Teaching Case, 311.

Nakatani, T., Tsumaki, T., 2011. Requirements maturation analysis based on the distance between the source and developers. Proceedings of the CHASE. pp. 88–91.

Nilson, L., Stranny, C., 2014. Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time. Stylus Publishing, Sterling, VA.

Oz, E., 1994. When professional standards are lax: the CONFIRM failure and its lessons. Commun. ACM 37 (10), 29–43.

Palomares, C., Quer, C., Franch, X., Renault, S., Guerlain, C., 2013. A catalogue of functional software requirement patterns for the domain of content management systems. Proceedings of the SAC. pp. 1260–1265.

Preece, J., Rogers, Y., Sharp, H., 2006. Interaction Design: Beyond Human-computer Interaction, 1st edition. Wiley.

Preece, J., Rogers, Y., Sharp, H., 2015. Interaction Design: Beyond Human-computer Interaction, 4th edition. Wiley.

Proynova, R., Paech, B., Koch, S.H., Wicht, A., Wetter, T., 2011. Investigating the influence of personal values on requirements for health care information systems. Proceedings of the SEHC. pp. 48–55.

Quintanilla Portugal, R.L., Engiel, P., Pivatelli, J., do Prado Leite, J.C.S., 2016. Facing the challenges of teaching requirements engineering. Proceedings of the ICSE. pp. 461–470.

Radermacher, A., Walia, G., Knudson, D., 2014. Investigating the skill gap between graduating students and industry expectations. Proceedings of the ICSE Companion. pp. 291–300.

Rempel, G., 2015. Defining standards for web page performance in business applications. Proceedings of the ICPE. pp. 245–252.

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

Röder, H., 2011. A pattern approach to specifying usability features in use cases. Proceedings of the PEICS. pp. 12–15.

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

Mohan, S., Chenoweth, S., 2011. Csse371 project rubrics. URL https://tinyurl.com/y7xgd95h.

Saiedian, H., Dale, R., 2000. Requirements engineering: making the connection between the software developer and customer. Inf. Softw. Technol. 42 (6), 419–428.

Scepanovic, S., Beus-Dukic, L., 2015. Teaching requirements engineering: Euroweb experience. Proceedings of the ECSAW. pp. 38:1–38:5.

Schneidewind, L., Hörold, S., Mayas, C., Krömker, H., Falke, S., Pucklitsch, T., 2012. How personas support requirements engineering. Proceedings of the UsARE. pp. 1–5.

Sedelmaier, Y., Landes, D., 2017. Experiences in teaching and learning requirements engineering on a sound didactical basis. Proceedings of the ITiCSE.

Shaban-Nejad, A., Haarslev, V., 2007. Towards a framework for requirement change management in healthcare software applications. Proceedings of the OOPSLA. pp. 807–808.

Shaw, M., Herbsleb, J., Ozkaya, I., 2005. Deciding what to design: closing a gap in software engineering education. Proceedings of the ICSE. pp. 607–608.

da Silva, A.R., Savić, D., Vlajić, S., Antović, I., Lazarević, S., Stanojević, V., Milić, M., 2015. A pattern language for use cases specification. Proceedings of the EuroPLoP. pp. 8:1–8:18.

Song, X., Hwong, B., Matos, G., Rudorfer, A., Nelson, C., Han, M., Girenkov, A., 2006. Understanding requirements for computer-aided healthcare workflows: Experiences and challenges. Proceedings of the ICSE. pp. 930–934.

Souza, V.E.S., Lapouchnian, A., Mylopoulos, J., 2012. (Requirement) evolution requirements for adaptive systems. Proceedings of the SEAMS. pp. 155–164.

Svahnberg, M., Aurum, A., Wohlin, C., 2008. Using students as subjects - an empirical evaluation. Proceedings of the ESEM. pp. 288–290.

Withall, S., 2007. Software Requirement Patterns, 1st edition. Microsoft Press.

Woolcock, M.J.V., 2006Woolcock. Constructing a syllabus. Brown University, Harriet W. Sheridan Center for Teaching and Learning, Providence, RI.

Chandan Rupakheti is an Associate Professor in the Computer Science and Software Engineering department at Rose-Hulman Institute of Technology. Chandan's teaching and research interests lie in areas of Software Engineering such as Program Analysis, Software Architecture and Design, and DevOps. Chandan is also a practicing architect and does consulting works for industries in Indianapolis and Chicago areas. At Rose-Hulman, Chandan has worked extensively in revamping Software Engineering courses such as Requirements Engineering, Software Design, Software Architecture, and Formal Methods.

Mark Hays is an Assistant Professor of Computer Science and Software Engineering at Rose-Hulman Institute of Technology. Prior to joining Rose-Hulman, he worked for IBM Software Group for eight years. His work emphasizes computer-aided software engineering, particularly to automate common software testing, debugging, and requirements tracing tasks. While the shifting software landscape requires him to regularly re-evaluate his favorite CASE tools, his love for Chicago-style pizza remains timeless.

Sriram Mohan is an Associate Professor in the Computer Science and Software Engineering Department at Rose-Hulman. During his time at Rose-Hulman, Sriram has served as a consultant in Hadoop and NoSQL systems and has helped a variety of clients in the Media, Insurance, and Telecommunication sectors. Sriram maintains an active research profile in data science and education research that has led to over 30 publications or presentations. At Rose-Hulman, Sriram has focused on incorporating reflection, and problem based learning activities in the Software Engineering curriculum. Sriram has led the revamp of the entire software engineering program at Rose-Hulman.

Stephen Chenoweth Steve came to Rose-Hulman from a career working for computer and telecom vendors. This included doing requirements and systems engineering, software development, project and resource management, and internal consulting. Throughout that career, he also taught and worked on educational projects, including online education. At NCR, Steve was one of four architects who designed the first point of sale system using peer-to-peer networks of PC-based machines. At AT&T and Lucent,

Steve taught systems architecture, did architectural consulting and led architecture reviews of new systems. At Rose-Hulman Steve is active in promoting academic change and working to match educational programs to the needs of software professionals over their careers.

Amanda Stouder graduated from Rose-Hulman Institute of Technology with a Bachelor's Degree in Computer Science and Software Engineering. Amanda's career has centered on custom software development, with a focus on requirements engineering and verification testing, including working in the FDA regulated space on projects supporting multiple clients with differing needs. Amanda chose to pursue teaching as a Visiting Professor of CSSE at Rose-Hulman while continuing consulting work through her own company. She has continued to do requirements, software development, and testing through her company while teaching these concepts, now as an Assistant Professor for the Practice of CSSE.