New Orleans JAZZED about Engineering Education

Paper ID #16525

A Merged Software Requirements and Architecture Course

Dr. J. Scott Hawker, Rochester Institute of Technology (COE)

Dr. Hawker graduated with a B.S. and M.S. in Electrical Engineering from Texas Tech University in Lubbock, Texas, in 1981 and 1982, respectively. He graduated with a Ph.D. in Electrical Engineering from Lehigh University in Bethlehem, Pennsylvania, in 1990. He has over 15 years of industry experience developing large-scale, multi-agent information and control systems for diverse applications including manufacturing, combat pilot decision support and mission management, robotics, and surveillance. In these areas, he developed and applied technologies including distributed, component-based software architectures, software and systems engineering process models, intelligent control, the semantic web, and real-time artificial intelligence. In 1999, Dr. Hawker joined the Computer Science Department at the University of Alabama as an Assistant Professor focusing on software engineering, and in 2004 he moved to the Software Engineering Department at RIT. Dr. Hawker is also co-director of the Laboratory for Environmental Computing and Decision Making, which focuses on modeling and understanding the impact of freight transportation and automotive industry activities and public policies. Dr. Hawker also performs research on improved software development processes for scientific software. Dr. Hawker is a member of the ASEE, IEEE, IEEE Computer Society.

Mr. Robert Kuehl, Rochester Institute of Technology

Robert is an adjunct lecturer in the Software Engineering Department at the Rochester Institute of Technology. He has primarily been teaching undergraduate courses in Requirements Engineering and Software Architecture Design, and Human Computer Interface Design. Prior to teaching Robert worked in industry at the Eastman Kodak Company in a variety of software engineering related roles in support of digital imaging product research and development.

Mehdi Mirakhorli,

Dr. Mehdi Mirakhorli is an Assistant Professor at Rochester Institute of Technology with a research background in software architecture design, requirements engineering, and application of data mining in software engineering. Previously, he worked as a software architect on large data-intensive software systems in the banking, meteorological and health care domains. He has served on the Program Committees for several conferences and as Guest Editor for a special edition of IEEE Software on the Twin Peaks of Requirements and Architecture. Dr. Mirakhorli has received two ACM SIGSOFT Distinguished Paper Awards at the International Conference on Software Engineering.

A Merged Software Requirements and Architecture Course

J. Scott Hawker
Robert Kuehl
Mehdi Mirakhorli
Rochester Institute of Technology
Rochester, New York, USA

Introduction

The undergraduate software engineering curriculum at the Rochester Institute of Technology used to have a course in Software Requirements Engineering, and a separate course in Software Architecture Design. As part of a curriculum re-design, we merged the requirements and architecture courses to make room for courses in software security and web applications. Overall, the students benefit from the added courses in the curriculum, but merging the requirements and architecture courses was a challenging process since some material from the two courses needed to be deleted. However, merging the two courses was a valuable process in that it brought the two strongly-related topics of requirements and architecture together. Practitioners in industry do not treat requirements and architecture as isolated practices. Merging the two courses helps students learn ways of separating, yet bridging, the gap between problem-space thinking (requirements) and solution-space thinking (design).

This paper discusses the practical motivations to merge these two courses, the valuable and challenging decisions in merging them, and lessons learned after teaching the merged course. In searching the software engineering education literature, we found topics covering requirements and architecture separately, but we were most influenced by literature that merged the two closely-related topics.[1,2]

Drivers of Course Change

The main event that triggered the requirements and architecture course merge was a university-wide decision to change the academic calendar from a quarter-based system to a more traditional semester-based system. This gave us the opportunity to holistically update the curriculum. We added a software security course and a web-based design course, in response to feedback from our Industrial Advisory Board. To accommodate these important new courses and meet state-imposed constraints and accreditation requirements, we had to re-allocate contact hours across the curriculum.

One of the opportunities in meeting the new curriculum requirements was merging the requirements and architecture courses. By merging the two courses, we could provide course team projects and individual activities that spanned the requirements specification and architecture design activities of software development. The tight relationship between

requirements and architecture development is often described with the Twin Peaks model,[1,2] emphasizing the iterative co-development of requirements and architecture. When the courses were separate, we could not have a single project that spanned the two courses. With the merged course, the students carry the same project from inception through to requirements specification and architecture design. This helps the students learn how requirements, especially non-functional quality requirements, drive architecture design decisions.

In a recent Twin Peaks workshop[3], participants found that "a major shortcoming was identified in that requirements and architectures are often taught independently and in a fashion that resembles a waterfall process." Our merged requirements and architecture course addresses this shortcoming.

Another driver of change impacting the merged requirements and architecture course was to make this course a "writing intensive" course in support of the general education requirements at our university. Given the document and model-centric nature of the course, the requirements and architecture course was a natural choice for a writing intensive course. However, that brought with it certain university-level requirements to address writing mechanics and style and to provide formative feedback opportunities before final document delivery. The new requirements and architecture course has an iterative structure for the team project which provides a number of opportunities for review and feedback.

The final, but most important, driver of the course merge was to address some of the difficulties of teaching software engineering undergraduate students how to think in the more abstract "big picture" systems perspectives of requirements and architecture. We have built into the course a number of opportunities to critically assess example artifacts and case studies that we have collected over the years. In addition, we have built in opportunities for students to specify, design, and assess the quality of their own work.

The merged requirements and architecture course leads into a two-semester capstone senior project. This senior project engages students and external sponsors in an end-to-end, requirements-to-delivery project. Requirements engineering and architecture design are central to this team-based senior project, and the merged requirements and architecture course gives students the system-wide problem-space/solution-space mindset necessary to succeed in their senior projects.

Course Structure

The merged requirements and architecture course is designed with three main threads that occur in parallel, as Figure 1 shows. We use a "just in time" studio lab philosophy and a project-based course structure. A typical class session involves a lecture and either an individual activity or team-focused project time. We introduce students to a concept through reading assignments and

lectures, immediately give them a chance to apply that concept on an individual project, then apply that concept on a semester-long team project.

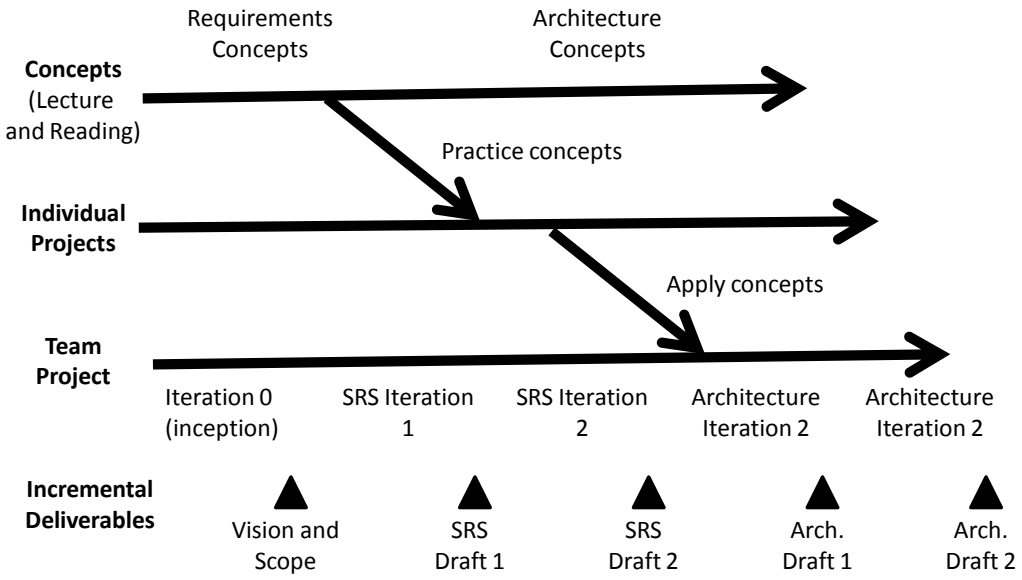

Figure 1. Overall Course Structure

Course Threads

1. Concepts (lectures and reading). The first one-third of the lecture and assigned reading are devoted to the requirements engineering activities of elicitation, analysis, specification, validation, and management (see Figure 2). For a textbook we use *Software Requirements* by Karl Wiegers and Joy Beatty[4]. As a transition to architecture topics, we focus on non-functional quality requirements and the importance of specifying them carefully, emphasizing that they be measurable. The remaining lectures and reading assignments address architecture topics, including architecture design tactics to use in response to quality attribute requirements, architecture styles, architecture analysis methods, architecture structures and document views, and interface (API) design. For this section of the course, we use the textbook *Software Architecture in Practice* by Len Bass, Paul Clements and Rick Kazman[5]

2. Individual Projects. There are a number of elements in the individual projects thread of the course. We assign reading and critical evaluation of existing requirements and architecture documents. We have a case study project where the students perform requirements analysis and architecture design steps for a given product description (an Internet TV). This allows them to apply the concepts from the lectures as individuals, first, before they apply the concepts on the team project. It is important that the instructor

provides quick feedback to correct misconceptions before they appear in the team project. Last, we ask the students to select a specific topic in requirements and architecture, research that topic by reading journal papers and conference proceedings, then write a short survey paper addressing that topic and present the topic in class.

3. Team Projects. The team project is the main focus of the course, emphasizing project-based learning. To begin, each student is asked to identify a possible project. The projects are often from family, friends, or work colleagues. Teams of four or five students are then formed at random, and the team selects one of the projects that the individual team members proposed. The project must have an external sponsor who serves as the customer and user stakeholder. We do not allow the students to create their own projects because they need to experience requirements elicitation from external stakeholders. The project sponsors agree to serve as customer stakeholders. Their main role is to engage in a discussion to provide and validate requirements. They are also invited to review the solution architecture as it evolves in response to the requirements.

   Once a sponsor and project topic have been selected, the student teams perform project inception ("iteration 0") activities, culminating in a Vision and Scope document that is validated with the stakeholders. The teams then perform two iterations of requirements engineering, delivering two releases of an evolving Software Requirements Specification (SRS). The first release draft SRS is an opportunity for formative feedback. The instructor performs a technical review of the document looking primarily for requirements completeness and clarity. Early drafts are notoriously ambiguous and incomplete. The instructor also provides feedback on document style and mechanics, in response to the university intensive-writing course requirements.

   Following completion of the SRS, the course lectures and activities guide the student teams in the transition from requirements to architecture design. There is an emphasis on identifying and refining non-functional quality attribute requirements as the basis for architecture design. The teams perform two iterations of architecture design and evaluation. They develop two drafts of an architecture design, with formative instructor feedback between drafts. During those iterations, students identify deficiencies in their requirements specifications and they have to cycle back and modify their SRS, keeping the project sponsor involved to validate requirements changes. At this point, the students are able to more easily distinguish and move between problem-space requirements thinking and solution-space design thinking. This exposes deficiencies in the requirements specification and opportunities in the architecture design.

   As with the SRS draft, the first iteration architecture document draft serves as a vehicle for formative feedback of technical content and document mechanics and style.

Throughout the requirements specification and architecture design activities the teams are encouraged to elicit direction and feedback from their external sponsors.

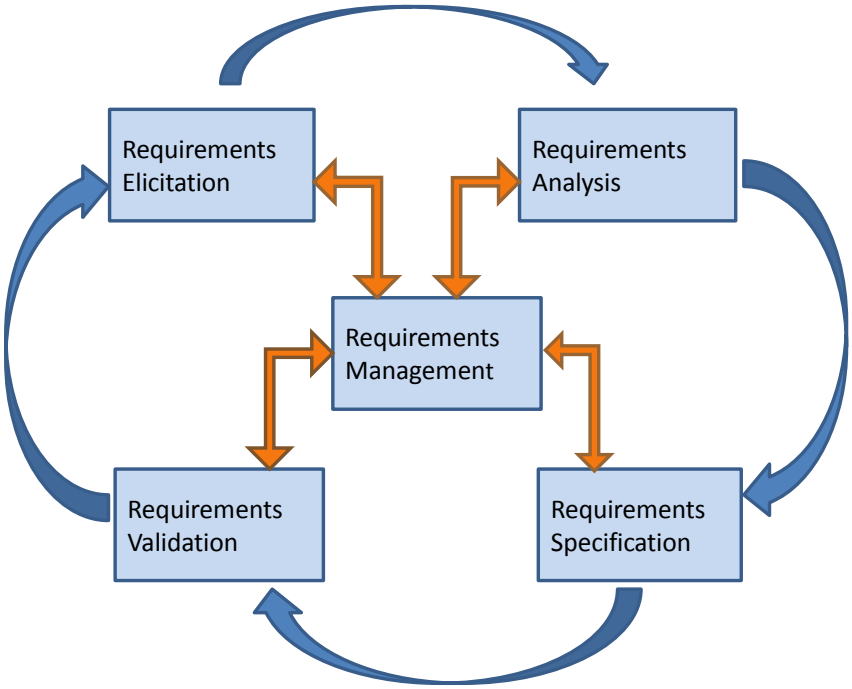

Figure 2. Requirements Engineering topics

Other Course Components

There are some additional course components that complement the three main threads.

- Rhetorical Theory in Requirements Elicitation. Interacting with external stakeholders for requirements elicitation is often difficult for students. We find that they are good with the facts and logic (*logos* in rhetorical theory), but lack empathy (*pathos*) and creating a sense of trust (*ethos*). We are fortunate to have a colleague who is trained in rhetoric and is a manager in a local information technology company. He presents a guest lecture on rhetoric concepts and how they apply in requirements elicitation.
- Architecture Design Workshop. One of the instructors who teaches this course has developed an architecture design workshop. Students are given an initial design of aircraft guidance, navigation, and control avionics. They then modify the architecture in response to additional quality attributes such as performance and availability.
- ATAM Workshop. The Architecture Trade-off Analysis Method (ATAM)[5] is one way to review the quality of an architecture design. As the architecture design iterations come to an end in the team project, we cross-assign members of the teams to review the

architectures. Each team sends two members to another team to act as reviewers, while the rest of the team presents and explains their architecture decisions with their assigned reviewers from another team. The analysis is guided by the architecturally significant quality attributes.

- Project Presentations. Each team gives three project presentations in the course. The first is a five-minute vision and scope statement. The second is a 15-minute presentation of the requirements (focusing on architecturally significant requirements). The last is a 15-minute presentation of the architecture design. Although there is not enough time for detailed discussion, it does expose the students to how other teams are approaching their requirements and architecture tasks.

What was Lost? What was Gained?

Given that we merged two courses into one, it was inevitable that some content was lost. However, there were gains in student's ability to think at the systems level of requirements and architecture.

The main loss was course content on the depth of requirements and architecture. We spend less time addressing requirements analysis techniques. We focus on modeling requirements with use cases with less emphasis on detailed "the system shall" statements. We don't spend as much time as we would like on architecture styles and patterns, on the use of design tactics to satisfy quality attribute requirements, and we have dropped content on architecture product lines and architecture reconstruction.

What we have gained, in return for losing course content, is a course which provides a more unified view of requirements and architecture with a focus on quality attributes that bridges the two system perspectives. Having the individual project, especially the Internet TV case study, provides an opportunity for quickly applying the course concepts and getting feedback before applying the course concepts to the team project. The team project gives the students an opportunity to work with external stakeholders beginning with project inception and ending with architecture design. This allows the students to experience the transition from problem-space software models (requirements) to solution-space software models (architecture designs) in an iterative, incremental process. It prepares our students to succeed in their Senior Design Capstone course where all these skills will need to come together in a year-long project.

Lessons Learned

We have run this course, with expected fine-tuning for a new course, for three academic years. The first year was as a pilot seminar under the quarter calendar, just before conversion to semesters. The compressed 10-week period of quarters made us think carefully about the breadth and depth of course content, and it allowed us to organize the course into the three concurrent threads in a just-in-time content delivery and practice.

The continuous and rapid instructor/student/team feedback in the three integrated threads is an important part of the course design. However, we have found that it causes a workload challenge for the instructor and the students. There is a lot of individual work to provide rapid feedback on, and the instructor becomes a reviewing stakeholder in each of the team projects – a critical point in the learning feedback cycle. To help manage this workload for both the instructor and the students, we are now performing the Internet TV case study in small, self-organizing teams.

Large classes of greater than 20 students certainly add to the workload problems, but they also make it difficult for the instructor to be available for deep interaction with teams in this project-based learning environment. Exploring and challenging requirements scope or architecture design decisions is an important part of project-based learning. There are also course schedule challenges. Not every project team can present their work to the class, or their presentation time is shortened, and not every individual can present their discoveries from their individual research topic activity.

Much of the grading evaluation is subjective. Especially at the level of abstraction of software system requirements and architecture, there are many possible interpretations of meaning and intent. Often it is difficult for the grader to understand what the students and teams were really thinking. Compounding this is the difficulty in assigning individual grades based on observation and peer evaluation of each individual's performance in team projects.

Given the sequencing of course topic delivery, the students still feel a "waterfall" nature to the process. We emphasize agile concepts such as incremental, iterative analysis and design and lightweight modeling – the Twin Peaks model. However, we have not been able to achieve multiple iterations between requirements and architecture as called for by the Twin Peaks model.

In some ways, we may have further confused the student's perceived boundary between requirements and design. We teach the students to build an object-oriented requirements analysis model to help validate the clarity and completeness of the requirements. The analysis model helps the students validate the requirements by viewing them through a feasible interaction of objects and subsystems – through a design lens. Although it is tempting for the students to take the analysis model and use it as a prototype design, we discourage that. Yes, the analysis model provides a first-draft decomposition of the system into functional modules, but we emphasize the importance of non-functional, quality requirements as key drivers of the architecture design. Although the student is not yet in a position to design an architecture, since that reading and lecture material has not yet been covered, they are beginning to perform design thinking in the context of requirements.

Coming into the course, students tend to underestimate the challenges and effort required to perform effective requirements engineering and software architecture design. This is in spite of the fact when questioned at the start of the semester, many students experienced poor requirements and architecture practices in their industry co-ops (completing a semester of co-op

is a pre-requisite to the course). Consequently there can be an initial tendency to produce quality-limited and quantity-limited artifacts at draft level. In most cases this self corrects following formative feedback and from individual activities to review existing requirements and architecture documents and case studies. This progression of incrementally improved quality in requirements and architecture design artifacts is one of the important learnings from the course.

Conclusion

In designing and delivering a Software System Requirements and Architecture course, we have experienced many of the trade-offs so common in software engineering. We have had to trade-off some breadth and depth of course content in order to strongly reinforce a key capability of software engineers: the ability to move smoothly between requirements-oriented problem-space models and design-oriented solution-space models and the ability to reason at multiple levels of abstraction.

Bibliography

1. Bashar Nuseibeh. "Weaving together requirements and architectures." *IEEE Computer* 34.3 (2001): 115-119.
2. Jane Cleland-Huang,., Hanmer, R. S., Supakkul, S., & Mirakhorli, M., "The twin peaks of requirements and architecture." *IEEE Software*, 30.2 (2013): 24-29.
3. Matthias Galster, Mehdi Mirakhorli *et.al.* "Views on Software Engineering from the Twin Peaks of Requirements and Architecture," ACM SIGSOFT Software Engineering Notes, September 2013 Volume 38 Number 5, pp. 40-42
4. Karl Wiegers and Joy Beatty, *Software Requirements*, 3rd Edition, Microsoft Press, 2013.
5. Len Bass, Paul Clements and Rick Kazman, *Software Architecture in Practice*, 3rd Edition, Addison-Wesley, 2013.