AC 2011-322: REUSE A ”SOFTWARE REUSE” COURSE

Nan Niu, Mississippi State University

Nan Niu is an Assistant Professor of Computer Science and Engineering at Mississippi State University. He received his Ph.D. in Computer Science in 2009 from the University of Toronto, where he specialized in requirements engineering for software product lines. His research interests include software reuse, requirements engineering, program comprehension, and software engineering education. He is a member of IEEE, IEEE Computer Society, and ASEE.

Dr. Donna Reese, Mississippi State University

Donna Reese is a professor and interim head of the Department of Computer Science & Engineering at Mississippi State University. She has been at Mississippi State since 1989 and served for 6 years as associate dean for the Bagley College of Engineering. She is a senior member of ACM and IEEE and was recently recognized by Tau Beta Pi with the McDonald Mentoring award.

Kui Xie, Mississippi State University

Kui Xie is an assistant professor of Instructional Technology at Mississippi State University. He received his Ph.D. in Instructional Psychology and Technology in 2006 from University of Oklahoma. His research interests include instructional design, computer-supported collaborative learning, motivation and cognition engagement in distance learning.

Chris Smith, PhD, PE, Mississippi State University

Dr. Smith is the Richard A. Rula Chair in Construction Engineering and Management and Assistant Professor in the Department of Civil and Environmental Engineering at Mississippi State University. He is a former U. S. Navy SEABEE Officer and has managed projects and programs worldwide. He has been accepted as an expert by the U. S. Court of Federal Contract Claims in the areas of Cost and Schedule. He is a former executive with Hill International and FTI Consulting and has been associated with the construction of Boston’s Central Artery Tunnel, Dubai Mall in the United Arab Emirates, and the U. S. Department of Energy’s Nuclear Waste Treatment Plant at Hanford, Washington.

Reuse a “Software Reuse” Course

Abstract

Software reuse is the use of existing software artifacts and knowledge to construct new software. Systematic reuse has always been a major goal in software engineering since it promises large gains in productivity, quality, and time-to-market reduction. One of the main reasons software reuse has not been systematically practiced is due to the lack of education: In a survey collected from 113 respondents from 29 organizations, primarily in the US, only 13% said they had learned about reuse in school[1].

This paper presents the creation of a graduate-level seminar course on software reuse in a US institution whose software engineering program aims to educate students with strong technical skills so they can start work as productive members on a software development team. Rather than reinventing the wheel in curriculum development, we adapted a software reuse course developed by Frakes at Virginia Tech[2].

This paper reviews the major challenges of software reuse education, describes the reuse of Frakes’ course modules and assessments, and discusses the modifications we made in our course. In particular, we modified our course by incorporating two pedagogical principles: active learning and cooperative learning. Redesigning the course from a lecture format to a seminar format allowed the students to play active roles in leading the classes and in discovering term paper topics that suited to their own research interests. Fostering collaborations among students and interactions between students and instructor allowed the students to recognize their individual accountability to the success of the group and the entire course. This paper reports instructor experiences, lessons learned, and recommendations for other educators considering the application of an active and cooperative learning approach for their software reuse courses.

Keywords: software reuse education; active learning; cooperative learning

Introduction

Software reuse is the process of creating software systems from existing software rather than building them from scratch[3]. The simple yet powerful vision of “not reinventing the wheel” has been successfully applied in manufacturing industries such as automobile and electronics. Reuse as a distinct field of study in *software engineering* is often traced to McIlroy’s paper that proposed basing the software industry on reusable components[4]. Note that McIlroy’s paper appeared in the 1968 NATO conference, which is generally considered the birthplace of the software engineering field. Therefore, from the beginning, software reuse has been touted as a means for overcoming the software crisis.

Software crisis was a term used in the early days of computing to describe the impact of the complexity of the problems which could be tackled. McIlroy felt that component libraries could contribute positively to writing correct, understandable, and verifiable computer programs[4]. Others have found software reuse is of interest because people want to build systems that are bigger and more complex, more reliable, less expensive and that are delivered on time[5]. In fact, the advantage of amortizing software development efforts through reuse continues to be widely acknowledged, even though the tools, methods, languages, and overall understanding of software engineering have changed significantly since 1968.

In spite of its promise, software reuse has failed to become standard practice for software construction[3]. Among the many causes of this failure, the lack of education is considered one of the most important. In a survey conducted in 1993 among 113 respondents from 29 organizations, primarily in the US, only 13% said they had learned about reuse in school[1]. Similarly, only 19% of respondents reported that they received training about software reuse at work[1]. Simply put, those who were not trained in software reuse were unlikely to practice it. The situation remained largely unchanged. A recent software reuse status report[5] published in 2005 pointed out that reuse education is still relatively rare in both academia and industry, and that there has been little systematic study of how best to do reuse education.

In this paper, we present the creation of a graduate-level seminar course on software reuse in a US institution whose software engineering program aims to educate students with strong technical skills so they can start work as productive members on a software development team. Rather than reinventing the wheel in curriculum development, we adapted a software reuse course developed by Frakes at Virginia Tech[2]. In what follows, we review the major challenges of software reuse education, describe the reuse of Frakes' course modules and assessments, and discuss the modifications we made in our course. In particular, two pedagogical principles, active learning and cooperative learning, were incorporated in our course as a means of teaching software reuse more effectively.

Background

Software reuse has been practiced since programming began[5]. This certainly applies to students who have taken computer programming courses and/or written software applications. Software reuse can be practiced in various ways: copy and paste a textbook algorithm, copy and modify code snippets from the internet or a previous project, refactoring and re-modularization, instantiate a C++ template, create a library of components and modules, apply design patterns, define domain-specific languages (DSLs) and automate application generators, leverage software product line technologies by codifying a set of core assets and by engineering products based on the reusable assets, to name a few.

The idea of software reuse is simple, i.e., the use of existing software artifacts or knowledge to construct new software. We argue that one of the biggest challenges of software reuse lies in this simple vision, i.e., there must exist something in the first place, be it software artifacts or knowledge, for reuse to take place. This challenge also applies to reuse education, in that the student must have produced non-trivial software artifacts or accumulated practical software knowledge in order to appreciate the value of software reuse. This is the main reason that software reuse should be taught as a senior-level or graduate-level course after the students have gained sufficient software engineering background.

Even though reusable artifacts or knowledge exist, software reuse is often practiced in an *ad hoc* way. This is mainly due to the fact that the artifacts or knowledge that could be reused are not built explicitly with reuse as an objective. As a result, people need to spend significant effort in identifying reuse opportunities. Most students enrolling in a software reuse course may have already had this kind of *ad hoc* reuse experience, such as copy-and-paste small-scale code fragments. *Ad hoc* reuse offers very limited success because the reuse solution is for a specific problem or task that cannot be generalized to other situations. Therefore, another challenge in reuse education is to re-frame and transform the student's *ad hoc* reuse experience toward a *systematic* reuse approach. Here, systematic implies repeatable reuse practices and predictable reuse benefits. The application of a systematic approach is key to evolve software reuse into an engineering discipline.

Several reuse curriculum development efforts were made in the 1990's[2,6,7]. While the course[7] was a five-lesson section that was taught as part of a software maintenance course, each of the courses[2,6] was designed as a one-semester graduate-level seminar course. The target audience of these early courses was practicing software professionals, such as (fulltime) Ada programmers[6] or US Air Force's (future) software engineers[7]. The emphasis was placed primarily on specific code-level reuse techniques, such as structured programming[6] and library construction[7]. In contrast, Frakes' course taught at Virginia Tech[2] had covered a wider spectrum of reuse topics, ranging from economics and measurement to generative programming and re-engineering. The crucial aspect that distinguished Frakes' course from other reuse courses was probably the explicit introduction and teaching of *domain engineering*.

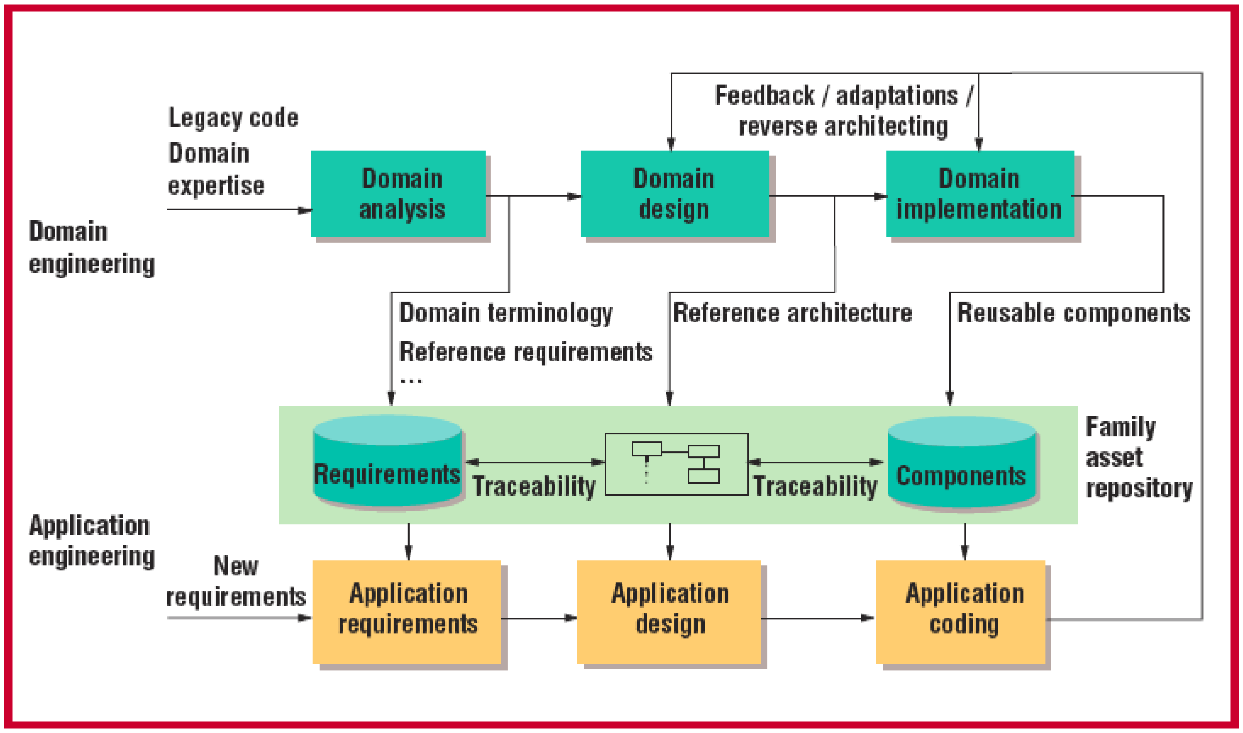


Figure 1. Domain engineering as a means of achieving systematic software reuse[8]

Domain engineering is the key to systematic software reuse. The basic insight is that most organizations build software systems within a few business lines, called domains, repeatedly building system variants within those domains. This insight can be leveraged to improve the quality and productivity of the software production process. Figure 1 shows the relationships between domain engineering and application engineering in the context of a software product line[8]. Domain engineering involves three phases: (1) *domain analysis* is the process of discovering and recording the commonalities and variabilities in a set of software systems; (2) *domain design* is to decide which platform components are needed; and (3) *domain implementation* is the use of the information from domain analysis and design to create reusable assets and new systems within a domain. Frakes' course[9] features the use of DARE (domain analysis and reuse environment)[10] and its online tool support[11] for the students to carry out a domain analysis project. The course has been evolved since 1993. It was last taught at Virginia Tech's Northern Virginia Center in spring 2010[9]. Table 1 summarizes the topics and assessments in its latest offering.

Table 1. Topics and assessments in Frakes' course: *Domain Engineering and Systematic Reuse*[9]

Topics	Assessments (*weight percentage*)
• Course Introduction • Domain Engineering • Product Line Engineering • Reuse Libraries • Reuse Design • Architectures • Programming Languages • Generative Methods • Measurement, Experimentation, Economics • Reliability and Safety • Re-engineering for Reuse	• Domain Engineering Project (*48%*) o Use DARE to perform domain analysis for one of the following domains: software metrics, conflation algorithms, or one of the student's choice o Use DAREonline to produce a domain book • Midterm (*30%*) • Term Paper (*10%*) o Create a research proposal (project, thesis, or dissertation) concerning domain engineering and software reuse • Presentations and Participation (*12%*)

Frakes' course is attractive for a number of reasons. First, it addresses the two major challenges of reuse education noted above: it teaches graduate students *advanced* topics in software engineering and it teaches domain engineering as a means of achieving *systematic* software reuse. Second, the domain engineering project of the course is both process-oriented (DAREonline[11] helps structure the steps involved in domain analysis) and product-oriented (the domain book outputted from DARE is a crucial component in the family asset repository). Finally, the course has a strong research component in that the term paper allows the students to relate the reuse course's learning experience to their own areas of interest. For these reasons, we decided to reuse Frakes' course to create a graduate-level seminar course on software reuse in our institution – a leading public research university in the US. While Frakes' course was advantageous in many aspects, we tried to incorporate two novel styles, active learning and cooperative learning, in our teaching of software reuse. Next, we present the pedagogical principles.

Theoretical Support

Many college teachers nowadays move past passive learning to *active learning* to find better ways of engaging students in the learning process. This means that instead of simply receiving information verbally and visually, students are receiving and participating and doing. Active learning derives from two basic assumptions: (1) that learning is by nature an active endeavor;

and (2) that different people learn in different ways[12]. It has been suggested that students who actively engage with the material are more likely to recall information[13]. In practice, active learning is often used as an umbrella term that refers to several instructional strategies. Examples of active learning activities include:

- A **short written exercise** is a good way to review materials and provide feedback.
- A **class discussion** requires the learners to think critically on the subject matter and use logic to evaluate their and others' positions.
- A **collaborative learning group** is where students are assigned in groups of 3-6 people and are given an assignment or task to work on together.
- A **class game** is an energetic way to learn because it not only helps the students to review the course material but it helps them to enjoy learning about a topic.

While other models of instruction exist to engage learners, it is important to note that lecture does have its place and that active learning should not be done without content or objectives. This requires students receive proper orientation and motivation, as well as constant guidance and feedback, from the instructor. The lectures given and other examples set by the instructor will play an even more important role in promoting active learning. In general, software engineering educators have reported positive experiences with active learning[14,15].

Cooperative learning is a type of active learning where the students work together in small groups to facilitate their own and the other members' learning. Millis[16] describes three premises underlying cooperative learning: (1) a respect for the individual differences among the students – intellectual, educational, social, and ethnic – and the belief that they all possess the potential to succeed in the class; (2) an active and constructive process that allows students to discovery and create their own knowledge; and (3) a social activity with a shared sense of community. Cooperative learning involves students working in teams toward a common goal. Elements of cooperative learning include:

- **Positive interdependence**: Team members rely on each other to achieve the common goal.
- **Face-to-face interaction**: Team members do most of the work together. They provide assistance, encouragement, and feedback to the other team members.
- **Individual accountability and personal responsibility**: Each team member is responsible for doing his/her share of the work, and is expected to master all necessary material.
- **Interpersonal and small-group skills**: Team members use effective communication and conflict-management skills.

- **Group processing**: Team members set common goals, reflect on team accomplishments, and make adjustments as necessary.

Since the first research study on cooperative learning in 1898, there have been nearly 700 relevant studies[17]. In software engineering education, for example, cooperative learning has been applied to teach software architecture in multiple-role teams[18]. The results show that, in general, cooperative learning leads to higher achievement and productivity by all students and deeper learning with longer retention. Our work extends the literature by sharing our experiences of incorporating active learning and cooperative learning activities in a software reuse course.

Method Details

We reused Frakes' course modules and assessments as a baseline and created a computer science special topics course, named "Software Reuse and Domain Engineering", in our institution. The institution is a public, comprehensive university that integrates research, learning, and service. The Computer Science and Engineering Department offers undergraduate degrees in both Computer Science and Software Engineering. At the graduate level, we offer Master of Science (M.Sc.) and Doctor of Philosophy (Ph.D.) degree programs in Computer Science.

The newly created course was offered in the spring 2010 semester. There were ten full-time computer science students in the course: seven were M.Sc. students and three were Ph.D. students. For the rest of this paper, we use pseudonyms and call the students: Alice, Bob, Chris, Dave, Eric, Frank, Grace, John, Mark, and Tom. Similar to Frakes' course, no mandatory textbook was required. The book, "Component Reuse in Software Engineering"[19], which could be accessed online, was optional. We relied on a series of research papers as the core readings for the course, which provided us with a better overview of recent research in software reuse. Our reuse course was designed to be a seminar course, not a pure lecturing course, at the outset. The active learning and cooperative learning strategies were implemented as follows.

➢ ORIENTATION AND MOTIVATION

The students could be active learners only if they were motivated and found the subject matter interesting and challenging. The first two classes of the semester were thus devoted to motivating the students. In particular, the importance of software reuse was emphasized. In addition to the commonly cited reuse benefits: reduced cost, assured quality, and increased productivity, the point of *"reuse is green"* was conveyed to the students. This point not only related to the historical concept of software crisis, but also tied up with the contemporary need of engineering environmentally friendly systems. The basic argument was: *If there's no reuse, it's a waste*. The analogy further expanded to education, i.e., the students constructed their own knowledge bases during the formal education, so that they could reuse (retrieve and adapt) the knowledge for the future.

Emphasis was also made for the students to realize that building things from scratch is difficult. This was partly achieved by quoting Carl Sagan: "If you want to make an apple pie from scratch, you must first create the universe." The students were then made aware that the majority of engineering tasks were not about radical design, but about normal design where different engineering solutions were studies and their tradeoffs were codified for reuse[20]. In software engineering, design patterns[21] were one of those attempts.

After the students realized that reuse was both necessary and beneficial, they were asked to share their software reuse experiences. It was not surprising that most students had only opportunistic reuse experiences, e.g., while getting ready to begin a new project, the team realized that there were similar components developed within the organization already. In contrast, few students had heard about software product lines or domain engineering. Examples of daily product lines were then brought to their attentions, such as Ford vehicles, Mac computers, and even McDonald sandwiches. The theoretical challenge of systematic reuse was summarized by the oracle hypothesis[5] concerning the ability to predict needed variabilities in future assets. The practical challenge of systematic reuse was illustrated by the domain engineering process (cf. Figure 1). The goal was to make it clear to the students that systematic reuse, though challenging, would mark a relatively mature engineering level of software development. The course was to have the students lead the study of the state of the art of software reuse and domain engineering, with the objective for them to advance the state of the practice of the field.

➢ SHORT WRITTEN EXERCISE

Recall that active learning should not be done without content. In a seminar-style course like ours, students must read the assigned reading(s) *before* the seminar and come to the class prepared. For each seminar, one required research paper and several background/optional readings were provided. We asked each student, except for the class discussion leader, to write an individual review of the required reading one day before the class day. To keep the paper review a *short* written exercise, we provided a template that included:

- A one-paragraph (3-5 sentences) summary of the paper;
- At least two bullet points that highlighted the student's "ah-ha" moments while reading the paper;
- At least two bullet-point questions that would provoke class discussion; and
- A free-form comments that had no length limit.

The "ah-ha" moments helped the students to record their favorite points, the least favorite points, insights, and other personal opinions about the paper. The students were also encouraged to record questions, confusions, and gaps in knowledge about the paper or the topic in general. These points were shared with the student who would lead the class discussion. The discussion leader found the reviews resourceful and was always willing to address several questions raised in the reviews in the seminar. This effectively promoted student interactions.

➢ LEARNING BY TEACHING

Each student was asked to lead part of the seminar on a self-selected topic. The ten topics are listed in Table 2. The topics were pre-determined by the instructor, and the required and background readings for each topic were provided right after the orientation class. The students had the opportunities to access all the readings and discuss interested topics with the instructor before selecting a topic to lead the class discussion. The topic selection was done on a first-come-first-serve basis. The students replied a group e-mail to achieve that. Alternatively, this could be done through postings in a newsgroup or an online bulletin board.

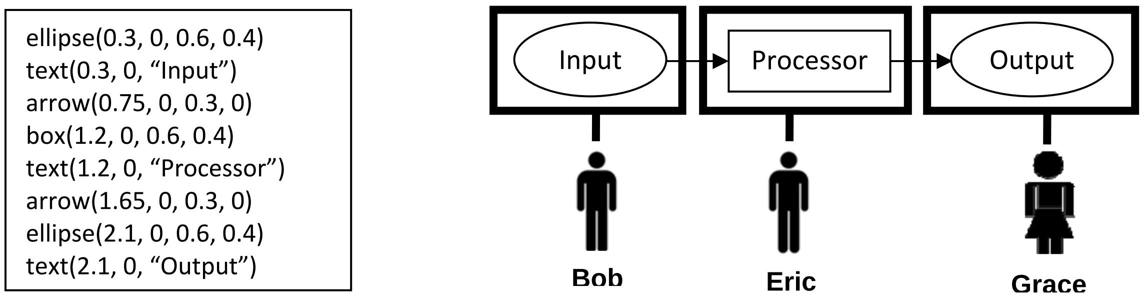
Table 2. Topics and assessments in our course: *Software Reuse and Domain Engineering*

Topics (*class discussion leader*)	**Assessments (*weight percentage*)**
• Course Orientation (*Instructor*) • Introduction and Motivation (*Instructor*) • Domain Engineering and DARE (*Instructor*) • Feature Orientation (*Eric*) • Requirements Reuse (*Mark*) • Design and Architecture Reuse (*Tom*) • Reuse Libraries (*Bob*) • Programming Languages (*Alice*) • Generative Methods (*John*) • Measurement and Experimentation (*Dave*) • Reuse Economics (*Chris*) • Re-engineering for Reuse (*Grace*) • Indexing and Retrieval (*Frank*) • Advanced Topic 1 o Systematic Literature Review (*Grace, Dave, John, Bob*) • Advanced Topic 2 o Software Ecosystems (*Tom, Mark, Eric*) • Advanced Topic 3 o Software Visualization and Reuse (*Chris, Frank, Alice*) • Course Evaluation and Summary (*Instructor*)	• Domain Engineering Project (*40%*) o Use DARE to perform domain analysis for one of the following domains: software metrics, conflation algorithms, or one of the student's choice o Use DAREonline to produce a domain book • Term Paper (*40%*) o A practical application of some method in software reuse and domain engineering o A critical review, e.g., a systematic literature review, of some aspects of software reuse o Encouraged to link the term paper with the student's thesis work • Presentations and Participation (*20%*) o Lead class discussion individually o Lead advanced topic as a group o Write paper reviews individually o Present term paper

Learning by teaching is an efficient instructional strategy that mixes guidance with active learning. It allows students to play active roles in the class by teaching new content to each other. In our reuse course, the student's leading discussion was regarded as an instance of learning by teaching for several of reasons. First, the student who led the class discussion was instructed *not* to give a presentation or a lecture of the paper. Instead, since the audience had already read the paper and submitted individual reviews, the discussion leader could provide a very succinct summary of the paper and spend more time in discussing thought-provoking topics. Second, most concerns addressed were raised by the audience. As noted above, the instructor shared with the discussion leader the suggested discussion points from individual reviews. This not only ensured a dynamic pool of discussion points from one topic to another, but also promoted the audience's active participation in the discussion. Finally, the discussion leader was responsible for only part of a class, typically the first half to two-thirds. The instructor would always follow up the discussion by further presenting supplementary materials, clarifying confusions, giving insights, and providing feedbacks to the discussion leader and the rest of the class.

➢ Class Game

One of the advantages of mixing active learning with learning by teaching is that students could choose their own methods and didactic approaches in leading the class discussion. Several students designed games to better engage the class. An example was the puzzle game designed by Alice. Alice led the discussion on “programming languages”, and the required reading was Jon Bently’s classic “programming pearls”[22] in which domain-specific languages (DSLs) were argued to be an effective reuse technique. Alice exploited the DSL, PIC[22] in designing a novel puzzle game. She randomly distributed a fragment of a picture to each student, and then showed, line by line, a program written in PIC via the projector. The PIC program was supposed to draw one component (e.g., a box, an ellipse, an arrow, etc.) at a time, and if the student believed the PIC program’s output was the component that had been distributed to him or her, the student would come to the board and stick his or her component in a proper position. If every student matched his or her component with the PIC program in the right order and position, a picture would be completed by the end of the game. Figure 2 illustrates PIC. The picture used in the puzzle game in the class was more complex and had more components than that of Figure 2.



```
ellipse(0.3, 0, 0.6, 0.4)
text(0.3, 0, “Input”)
arrow(0.75, 0, 0.3, 0)
box(1.2, 0, 0.6, 0.4)
text(1.2, 0, “Processor”)
arrow(1.65, 0, 0.3, 0)
ellipse(2.1, 0, 0.6, 0.4)
text(2.1, 0, “Output”)
```

Figure 2. Illustration of the puzzle game

➢ Roadmap

Figure 3 shows a roadmap after each of the ten students had led the discussion. The roadmap acted as a summary by emphasizing the students’ individual contributions across the domain engineering process. It provided the context and the big picture, so that the students could recognize their individual accountability to the success of the entire course. Note that the topics, measurement and experimentation led by Dave and reuse economics led by Chris, were so broad that they affected the whole picture. For Grace’s topic, re-engineering for reuse, an arrow/flow from application engineering to domain engineering was shown.

➢ Collaborative Learning Groups and Advanced Topics

As shown in Table 2, three advanced topics were discussed toward the end of our reuse course. These topics emerged either by students’ research area or by their choices of the term paper topic. The instructor assigned 3 to 4 students into each group. The group worked together to deliver a presentation to the entire class about the given topic. We felt such a collaborative learning group was an effective way to implement active learning because it caused the students to review the work at an earlier time to participate.

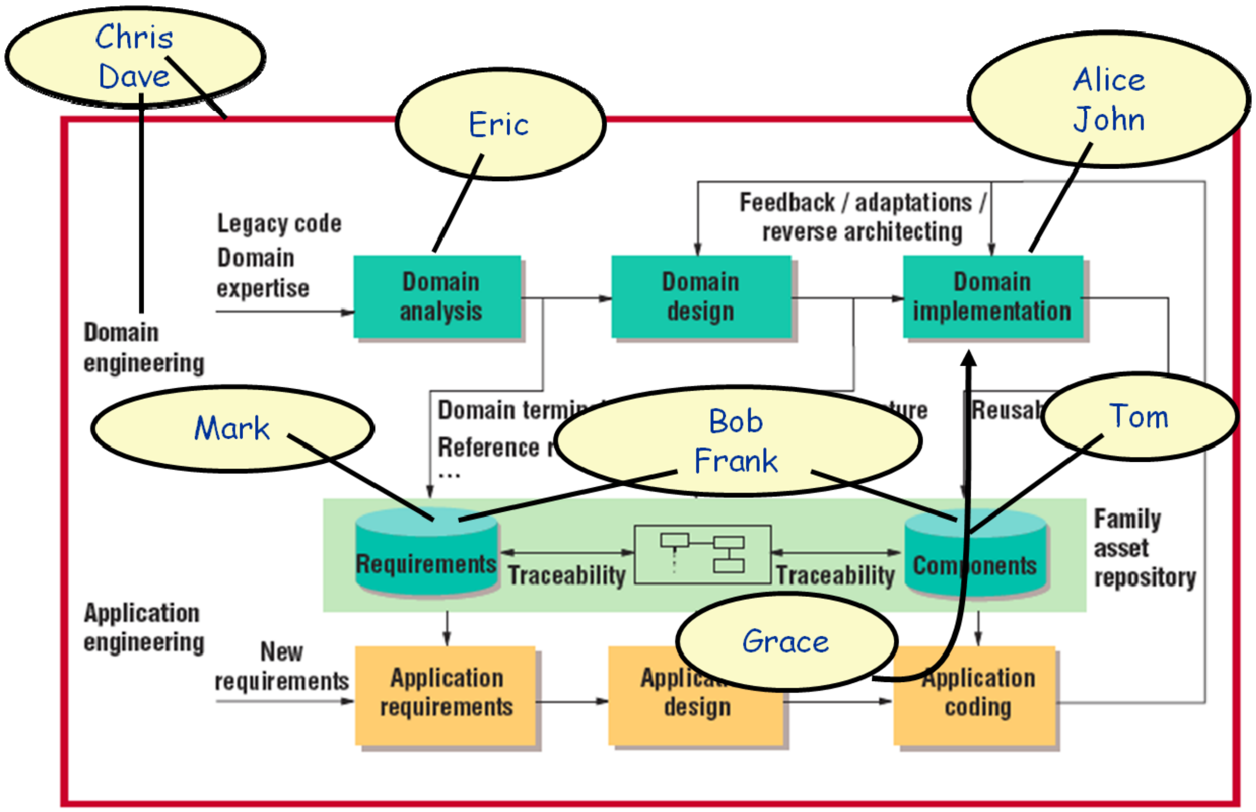


Figure 3. Domain engineering roadmap emphasizing individual contributions

Table 3. Term paper topic influenced by individually- or group-led topic in class

Student	Basic Topic Led	Advanced Topic Led	Term Paper Topic
Grace	(a) Re-engineering for Reuse	(k) Systematic Literature Review	(k) on “Reusability Metrics”
Dave	(b) Measurement and Experimentation		(k) on “Software Product Line Methodologies”
John	(c) Generative Methods		(k) on “Reuse Programming Languages”
Bob	(d) Reuse Libraries		(k) on “Reuse in Software Security Engineering”
Tom	(e) Design and Architecture Reuse	(l) Software Ecosystems	(l) on “Mobile Games Development”
Mark	(f) Requirements Reuse		(f) on “Functional Requirements Retrieval”
Eric	(g) Feature Orientation		(g) on “From E-Commerce to M-Commerce”
Chris	(h) Reuse Economics	(m) Software Visualization and Reuse	(m) on “Reuse Library Visualization”
Frank	(i) Indexing and Retrieval		(i) + (m) on “Visual Search”
Alice	(j) Programming Languages		“Software Reuse and Decision Making”

We felt that the workload of writing paper reviews should be reduced for the advanced topics, since each student had already written nine individual reviews thus far. As a result, the group had the freedom to choose required and background readings for the rest of the class, and no paper review was required. The instructor provided paper suggestions when needed. The group might assign no reading at all but rely on class presentation to introduce and discuss the topic. In fact, two of the groups assigned no readings before the class, but conveyed the seminar well in explaining the core ideas. Because games turned out to be very engaging in some of the individually led seminars, all the three groups adopted class games to make their seminars more interesting.

The collaborative learning groups not only offered the benefits of cooperative learning (positive interdependence, face-to-face interaction, individual accountability and personal responsibility, interpersonal and small-group skills, and group processing), but also helped the students to discover term paper topics that suited to their own research interests. A summary of each student's term paper topic, together with its relations to basic and advanced topics covered in the class, is given in Table 3.

Lessons Learned

A comparison between Table 1 and Table 2 highlights our revisions from Frakes' course. We combined "reuse design" and "architectures" into one seminar, split "measurement and experimentation" and "reuse economics" into two seminars, and skipped "reliability and safety" as one of the basic software reuse topics. We added "feature orientation", "requirements reuse", and "indexing and retrieval" to the basic topic's list, and further augmented the course with three advanced reuse topics. As for the assessment components, we dropped the midterm and emphasized more on class participation and the term paper that could potentially contribute to the graduate student's research.

As in Frakes' course, textbook was only optional in our course. Our students hardly read the textbook, mainly because reading and reviewing a dozen research papers were already time-consuming but also seemed sufficient. We felt that the status of software reuse textbooks could be greatly improved. On one hand, many books on software reuse were just collections of (outdated) readings. On the other hand, students gave negative ratings for the single-author or single-group authored textbooks[2].

Our course, in general, was successful. The students gave the course an overall rating of 5 (average=5, deviation=0.1) on a 5-point Likert scale (1 – negative end, 5 – positive end). We attributed the success to both the high quality of Frakes' course and our implementations of the active learning and cooperative learning strategies. We literally copied the domain analysis project from Frakes' course, because the project had good tool support and was both process- and product-oriented. The students liked the short paper review requirements that asked them to provide succinct bullet points.

We also identified several areas for improvement. First, some students would prefer to read a successful reuse story, e.g., a real-world case study, earlier in the course. Currently, one such paper[23] was assigned when "re-engineering for reuse" was discussed. The syllabus could be re-structured to accommodate the need. Moreover, a few students made the leading class discussion very much like a paper presentation. Other active learning methods, such as student debate, might be tried to improve the students' communication and presentation skills.

Summary

From the beginning of the software engineering field, reuse was recognized as a key enabler to overcome the software crisis. More than forty years later, software reuse is still seen as *potentially* a powerful means of improving software practice and productivity. The lack of reuse education has been a main factor that hinders the practitioners to systematically practice software reuse.

In this paper, we have presented our creation of a graduate-level seminar course on software reuse in a US institution. We reviewed the major challenges of reuse education and identified Frakes' course as a baseline to create our course. We described the reuse of Frakes' course modules and assessments, and also reported the modifications we made in our course. We incorporated active learning and cooperative learning in our course, and discussed our detailed implementations. Our experiences showed that: (1) Redesigning the course from a lecture format to a seminar format allowed the students to play active roles in leading the classes and in discovering term paper topics that suited to their own research interests; and (2) Fostering collaborations among students and interactions between students and instructor allowed the students to recognize their individual accountability to the success of the group and the entire course.

Recall Carl Sagan's quote that we used in the course orientation to motivate the class: "If you want to make an apple pie from scratch, you must first create the universe." Without Frakes' course, creating a software reuse course from scratch would be very difficult for us. Now with more creations of software reuse courses like ours, we hope to contribute to overcoming the difficulty to bring the seemingly simple idea of reuse to the forefront of software engineering. More importantly, by incorporating more effective and novel pedagogical principles and sharing the experiences of teaching software reuse, we hope to make the next generation of software practitioners realize software reuse's full potential.

Bibliography

[1] W. B. Frakes and C. J. Fox. Sixteen Questions about Software Reuse. *Communications of the ACM,* 38(6): 75-87, June 1995.

[2] W. B. Frakes. A Graduate Course on Software Reuse, Domain Analysis, and Re-engineering. In *Proceedings of the Sixth Annual Workshop for Institutionalizing Software Reuse*, Owego, NY, USA, November 1993.

[3] C. W. Krueger. Software Reuse. *ACM Computing Surveys*, 24(2): 131-183, June 1992.

[4] M. D. McIlroy. Mass Produced Software Components. In *Software Engineering*; *Report on a conference by the NATO Science Committee,* pp. 138-150, Garmisch, Germany, October 1968.

[5] W. B. Frakes and K. Kang. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering*, 31(7): 529-536, July 2005.

[6] Gertrude Levine. A Course in Software Reuse, Department of Computer Science, Fairleigh Dickinson University, Teaneck, New Jersey, 07666, http://www.umcs.maine.edu/~larry/latour/WISR/wisr4/proceedings/detex/levine.detex 1992.

[7] J. E. Cardow and W. D. Watson, Jr. A Practical Approach to Teaching Software Reuse. In *Proceedings of the Seventh SEI CSEE Conference (Software Engineering Education)*, A Course at Air Force Institute of Technology, Lecture Notes in Computer Science, Volume: 750, pp. 517-525, San Antonio, TX, USA, January 1994.

[8] F. van der Linden. Software Product Families in Europe: The Esaps & Café Projects. *IEEE Software*, 19(4): 41-49, July/August 2002.

[9] W. B. Frakes. CS 6704 – Advanced Topics in Software Engineering: Domain Engineering and Systematic Reuse, http://frakes.cs.vt.edu/6704S09.htm Virginia Tech's Northern Virginia Center, Fairfax, VA, spring 2010.

[10] W. B. Frakes, R. Prieto-Diaz, and C. Fox. DARE: Domain Analysis and Reuse Environment. *Annals of Software Engineering*, 5(1): 125-141, January 1998.

[11] R. F. D. Santos and W. B. Frakes. DAREonline: A Web-Based Domain Engineering Tool. In *Proceedings of the Eleventh International Conference on Software Reuse*, pp. 246-257, Falls Church, VA, USA, September 2009.

[12] C. Meyer and T. B. Jones. *Promoting Active Learning: Strategies for the College Classroom*, San Francisco: Jossey-Bass, 1993.

[13] J. S. Bruner. The act of discovery. *Harvard Educational Review*, 31(1): 21-32, 1961.

[14] S. Ludi. Active-Learning Activities that Introduce Students to Software Engineering Fundamentals, In *Proceedings of the Tenth Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 128-132, Caparica, Portugal, June 2005.

[15] S. Ludi, S. Natarajan, and T. Reichlmayr. An introductory software engineering course that facilitates active learning. In *Proceedings of the Thirty-Sixth SIGCSE Technical Symposium on Computer Science*, pp. 302-306, St. Louis, MO, USA, February 2005.

[16] B. Millis. Enhancing learning – and more! – through cooperative learning. Idea paper #38, The IDEA Center, 2002.

[17] D. Johnson, R. Johnson, and E. Holubec. *Cooperative Learning in the Classroom*, Alexandria, VA: Association for Supervision and Curriculum, 1994.

[18] S. Chenoweth, M. Ardis, and C. Dugas. Adapting Cooperative Learning to Teach Software Architecture in Multiple-Role Teams. In *Proceedings of ASEE Annual Conferences & Expositions*, Honolulu, HI, USA, June 2007.

[19] E. S. de Almeida *et al. Component Reuse in Software Engineering* (C.R.U.I.S.E), http://cruise.cesar.org.br/.

[20] W. G. Vincent. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*, The Johns Hopkins University Press, 1990.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[22] J. Bently. Programming Pearls: Little Languages. *Communications of the ACM*, 29(8): 711-721, August 1986.

[23] M. F. Dunn and J. C. Knight. Software Reuse in an Industrial Setting: A Case Study. In *Proceedings of the Thirteenth International Conference on Software Engineering*, pp. 329-338, Austin, TX, USA, May 1991.