Int. J. Engng Ed. Vol. 24, No. 4, pp. 659–670, 2008
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Addressing Diverse Needs through a Balance of Agile and Plan-driven Software Development Methodologies in the Core Software Engineering Course*

LUCAS LAYMAN
Department of Computer Science, North Carolina State University, Campus Box 8206, Raleigh, NC 27695, USA. E-mail: lucas.layman@ncsu.edu

LAURIE WILLIAMS
Department of Computer Science, North Carolina State University, Campus Box 8206, Raleigh, NC 27695, USA

KELLI SLATEN
Department of Mathematics and Statistics, University of North Carolina Wilmington, 601 S. College Rd., Wilmington, NC 28403, USA

SARAH BERENSON
Department of Curriculum and Instruction, University of North Carolina Greensboro, Curry Building, PO Box 26170, Greensboro, NC27402, USA

MLADEN VOUK
Department of Computer Science, North Carolina State University, Campus Box 8206, Raleigh, NC 27695, USA

The software industry uses a mixture of plan-driven and agile techniques, and educators must prepare students for industry needs while creating an effective educational environment that appeals to a diverse student population. We describe the undergraduate course in software engineering at North Carolina State University, which teaches both agile and plan-driven practices while emphasizing collaborative and active learning. We present demographics, personality types, and learning styles from 400 students, and provide statistical analyses and student testimonials on the impact of our course. Students have reacted favorably to the course and are better prepared to meet the diverse needs of industry.

Keywords: software engineering education; agile methods; personality types; learning styles

INTRODUCTION

EDUCATORS HAVE RESPONSIBILITIES to both prepare students for their future careers and to provide inclusive instruction for all students. The current landscape of the software industry and student enrollment in the computer science discipline particularly exaggerates these responsibilities. The skills desired of workers in the software development industry have changed as organizations diversify their practices and processes from traditional plan-driven [1] software development to include newer agile [2] software methodologies. Additionally, software engineering is projected to be one of the fastest growing occupations in 2004–14 [3]. Yet 2006 statistics show only a slight increase in computer science enrollment following a 50% decline over three years [4], with only 15% of undergraduate computer science majors being women and ethnic minorities (African–American, Hispanic, Native American) [4]. Today's educators must strive to prepare their students for the increasingly diverse needs of industry while continuing to make computer science education inclusive and appealing to a new generation of diverse students.

In this paper, we describe the undergraduate software engineering course at North Carolina State University (NCSU)†. The course prepares students for the current demands of industry with

* Accepted 25 April 2008.

† http://www.csc.ncsu.edu/courses/undergrad/index.php

a universal design for software engineering education meant to address the diverse pedagogical needs of students, regardless of gender, ethnicity, age, learning style, or personality type. The course, part of our core curriculum, educates students in both agile and plan-driven practices and in composing an appropriate development methodology tailored to the current project and team. Elements of our course, including an emphasis on collaboration, active learning, and practical software projects, allow us to more effectively attract, reach, and retain students, in particular Millennials [28, 30, 42], women, and minorities. We have collected the personality type and learning style information of 400 students in our course over several years. Statistical analysis of student grades in our course have shown no difference in the performance of different personality types and learning styles, whereas previous studies of engineering and computer science courses have shown disparities in the performance of particular groups [5–9].

In this paper, we discuss the changing needs and expectations that industry has of today's students. We then describe the layout, progression, and facilities of the undergraduate software engineering course at NCSU with specific examples of assignments and course projects. We discuss how our course appeals to the specific learning needs of the Millennial generation, women, ethnic minorities, and a variety of personality types and learning styles encountered in today's students. Finally, we provide analysis of student performance in our course and an overview of student testimonials about the course. From the example of our course, we hope to inspire other educators to evolve and/or apply our method for preparing students for their careers in software development.

DIVERSE NEEDS OF INDUSTRY

Software development, when described by process models, can be characterized as plan-driven [1] or agile [10]. The plan-driven models, such as the Waterfall process model [11] and the Spiral model [12], have an implicit assumption that much information about the product being developed can be obtained up front. The overriding philosophy is that the cost of product development can be minimized by creating detailed plans and by constructing and inspecting architecture and design documents to minimize risk prior to initiating code development. Teams often spend considerable effort in the planning stages before code implementation begins.

Alternately, agile models are considered to be best suited for projects in which a significant amount of change is anticipated due to unknown and evolving requirements [2, 10, 13]. Agile models have existed for some time in the form of iterative and prototyping process models [14]. Some more recent examples are Extreme Programming (XP) [15], Scrum [16], Crystal [2], and Feature Driven Development [17]. The philosophy with agile methods is that spending considerable effort to create a detailed plan may not be worthwhile due to the inevitability of change. Spending significant amounts of time creating and inspecting an architecture and detailed design for the whole project is similarly not advisable. Instead, agile software developers spend time planning and gathering requirements for small iterations throughout the entire lifecycle of the project.

Plan-driven methodologies have been used for many years in industry. Plan-driven methodologies are most often taught in universities as the formal process for creating software systems in industry due to this long history and in part because the stepwise nature of the Waterfall model provides a natural sequence of instruction in the software engineering course. Agile methods became more popular in the late 1990s. In the early years, many practitioners, researchers, and academics were skeptical of the ability of agile methodologies to guide the development of high quality, maintainable products. However, the use of agile practices and methodologies has rapidly grown over the last decade. A recent survey of 780 practitioners indicates that 69% of organizations have adopted agile techniques in some capacity [18], and recent conferences on agile methodologies have been filled to capacity by attendees from a wide range of commercial and government software organizations.

In light of increasingly balanced adoption of agile and plan-driven methods, the current generation of students needs hands-on practice with agile practices and methodologies prior to joining the workforce but not at the neglect of their knowledge of plan-driven methodologies. Furthermore, companies have long stressed the importance that students graduate with communication and teamwork skills. A strong set of skills in both the plan-driven and agile traditions and the training to judiciously apply the appropriate techniques for the given software project and team are essential for development of high-quality, large-scale software.

COURSE DETAILS

The overall goal of the software engineering course at NCSU is to teach students practical techniques, tools, and processes that they will encounter in professional software development. In this section, we describe the general layout of the course as well as the specific phases and learning objectives emphasized during the course.

Course layout

At NCSU, the software engineering course is a required course in the core curriculum and is taken by students in their third or fourth year of undergraduate study. As is the case with many computer

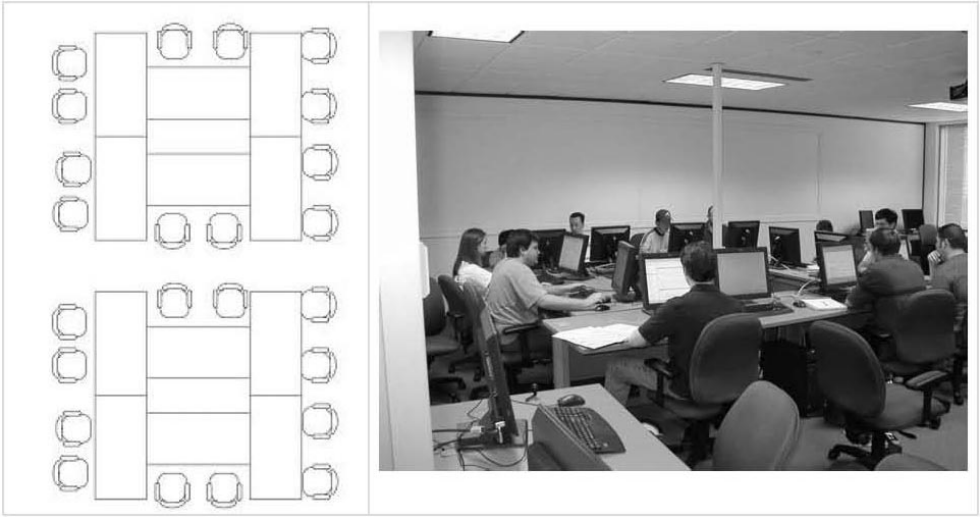

Fig. 1. Software engineering lab.

science curricula, our core curriculum has only one required software engineering course. Each semester, the course has an average of 50–70 students. Two, 50-minute lectures and one, two-hour closed lab session take place each week. The lab sessions are conducted in a room with computers (shown in Fig. 1), taught by graduate teaching assistants (TAs), and have 20–24 students per session.

The lecture sessions typically cover concepts and theories, such as discussions of software processes and testing strategies. Lecture sessions are punctuated by frequent brainstorming sessions where students discuss a question or complete a small task with their neighbors. Aside from encouraging the students to become active (rather than sit and listen for 50 minutes), these quick collaborations gives the students the opportunity to ask each other questions on the lecture material. If a student has trouble understanding a concept, then maybe his or her peers can help. If other students do not understand, then the knowledge that others are confused as well gives them the courage to ask the instructor for clarification or examples.

In the weekly two-hour lab sessions, the students receive hands-on experience with the concepts taught in lecture. In the lab sessions, students participate in project planning, learn to use components of the Eclipse development environment*, become familiar with testing tools such as JUnit†, write requirements documents, create software designs, and participate in other hands-on exercises. The physical arrangement of the lab space is non-traditional. The tables are arranged in rectangles so that students are facing each other (see Fig. 1), facilitating team communication. Each computer in the lab is equipped for pair programming [19] whereby two students work jointly on one computer, collaborating on the same design, code, or test. All computers in the lab have two monitors, two mice, and two keyboards.

Resources for the students are posted on the OpenSeminar in Software Engineering [20] (http://openseminar.org/se/) and are freely available to the public. These resources include lab activities, tutorials, and three Java testbeds to aid in active learning:

- CoffeeMaker is a small command-line application;
- RealEstate is a medium size GUI application; and
- iTrust‡ is a relatively large web-based, healthcare application that utilizes a database.

A variety of software engineering artifacts are available for each of these three testbeds, including requirements documents, automated JUnit and FIT§ test cases, and UML class and sequence diagrams. The details of the course, including the introductory readings, lecture slides (some narrated via the Camtasia tool), syllabus, lectures, and assignments, can be found on OpenSeminar.

Attributes of the semester project

Throughout the semester, the students enhance the functionality of a software application. This application carries through all homework assignments and the team project. To help create an environment reflective of industry, we create projects that have several attributes: *social relevance/practicality, security implications, privacy concerns* and *maintenance/enhancement experience.*

We focus on making our assignments meaningful, *practical and socially relevant* [21]. For example, the course projects in recent years have included a system for managing the assignments of social work students to community organizations,

* http://www.eclipse.org

† http://www.junit.org. JUnit is an open source, white-box unit testing framework.

‡ http://agile.csc.ncsu.edu/iTrust/

§ http://fit.c2.com. FIT is an open source framework for automating functional and/or acceptance tests.

a web portal for managing and disseminating data collected from a state forest, and a role-based healthcare application.

The students are taught to build *security* into their software [22, 23] via software engineering practices. Typically, the semester project is a web-based application. Web-based applications are common in today's industry and provide the ideal development environment to demonstrate security vulnerabilities and attacks such as SQL injection. Students also learn how to prevent such security vulnerabilities through input validation, information hiding, and the use of prepared statements.

Privacy concerns are some of the most pervasive issues in software development today. Students must learn the importance of developing software that is compliant with privacy legislation and business privacy policy. The iTrust health-care project used in our course has significant privacy concerns. The students analyze the iTrust requirements for compliance with Health Insurance Portability and Accountability Act of 1996 (HIPAA*) and then implement or enhance functionality with role-based access consistent with the associated privacy requirements.

Finally, the projects used in our course focus on *maintenance and enhancement*, rather than implementing a new project from scratch. Most students who enter industry will take on a testing or maintenance role, rather than be put on a 'Greenfield' project (indeed, very few developers anywhere work on exclusively new code). The projects we provide to the students contain adequate JUnit and FIT automated tests to maintain the integrity of the existing system as it is modified. The provided tests also demonstrate to the students the necessity of regression testing as the students will inevitably break an existing test while enhancing or refactoring existing functionality.

The first nine weeks: Software engineering practices

The course exposes students to a range of software development practices that span both agile and plan-driven software development processes. During the first nine weeks of the class, students learn software development practices in a *process-neutral* way. For example, students learn three different ways to document requirements: the traditional 'the system shall' approach, use cases/features, and user stories. The class discusses the pros and cons of each of the three forms by considering the time investment, thoroughness, and specificity of each, and the characteristics of the project/team for which each of the three types might be appropriate. Students learn about requirements, design, and testing principles, including specific practices such as pair programming, inspections, white box testing, configuration management, and more. By teaching the students a variety of tools and discussing the pros and cons of each in a process-neutral way, we give the students the basis for deciding which tools and practices to apply in a given scenario.

Testing is the first subject taught in the class, and the students learn a variety of black-box and white-box strategies and tools. By teaching testing first, the students are provided with a gentle introduction to the application they will be enhancing throughout the semester. The students learn about the operational behavior of the system through black-box tests of the system where they are challenged to act as a normal user and find bugs in the behavior of the system. The students are then introduced to the architecture, design, and control flow of the system through white box testing exercises. After testing is introduced, the other software development practices are introduced in an order commensurate with the Waterfall model.

During the first nine weeks of the class, the students are given four homework assignments lasting one to three weeks each. Each of these assignments builds upon the previous assignment and leads into the team project.

Assignment 1—Personal Web Page. The first assignment is to create a personal webpage that is linked from a password-protected class webpage. Students can learn about their partners and teammates and view their schedules by looking at these web pages.

Assignment 2—Testing Focus. This assignment is completed in pairs. The TAs assign pairs within the lab sections, respecting students' input on who they do not wish to work with, but not specifically allowing students to choose their partners. In the laboratory, the students are shown for the first time the application they will be enhancing throughout the semester. The students are given the application to test, the requirements for the application, a black box test plan, and extensive JUnit and FIT tests for the application. Prior to this assignment, students learned about black box exploratory testing [24] in lecture. All pairs perform testing on the application and report failures via the Bugzilla† bug tracking tool. The failures are not intentionally injected but are inevitably found by the students. The assignment requires the students to write a white-box JUnit *and* black-box FIT test to reveal each of the ten defects. The students must then fix the implementation, thereby passing the new JUnit and FIT tests.

Assignment 3—Design Focus. In Assignment 3, the students enhance the application by implementing new functionality using one of the design patterns [25] that has been discussed in lecture. The third assignment is also done in assigned pairs. As such, the students need to decide whose Assignment 2

* http://www.hhs.gov/ocr/hipaa/

† http://www.bugzilla.org/

project to enhance based upon each student's confidence in his or her previous project. Having two projects to choose from also aids in the case where a student may have had problems completing the prior assignment. The need to continue to enhance a product is a painful lesson for some students but also a lesson in the realities of industrial software development. The two-week assignment progresses in a mini-waterfall methodology as follows:

- Week 1: Students are given requirements statements and must turn in a black box test plan, FIT tables which outline the automation of at least three test cases/requirements, and a class diagram of their design, which demonstrates the use of the specified design pattern.
- Week 1 laboratory: Students exchange designs with another pair and peer critique each other's design. The lab TA chooses a particularly good design and reviews its positive attributes with the lab group.
- Week 2: Students turn in complete implementation of the new requirement, including 80% JUnit statement coverage and a minimum of three automated FIT tests per requirement.

Assignment 4—More Design Focus. The final homework assignment is completed individually by the students and follows the same weekly layout as Assignment 3. The students further enhance the same application but, for this assignment, the requirements are in the form of use cases [26] and students are required to turn in a UML sequence diagram for each use case. The scope of the requirements is scaled back since students are working individually instead of in pairs. This assignment gives the teaching staff the ability to review the students' performance on a programming assignment when working alone.

The transition week: Composing practices into methodologies

After learning software development practices in the first nine weeks, several lectures focus on how these practices are grouped in different ways to form software processes. Utilizing one set of practices in a particular way results in a traditional Waterfall model; using a different set of practices composes Extreme Programming (XP) [15]; yet another different set of practices comprises the Team Software Process (TSP) [28], and so forth. The students learn that the practices are the building blocks for the processes, and the best process for a particular project and team should be chosen based upon the specifics of the project and the context and demographics of the team. Specifically, the students learn to choose between an agile methodology, a plan-driven methodology, or the creation of a hybrid methodology using a risk-based approach purported by Boehm and Turner [29].

The last six weeks: Team project

During the last six weeks of the semester, the students work in teams to further enhance the semester project in more significant ways. The students are placed in groups of four or five and are given a requirements specification, access to a version control system, and any other technologies they may need. As with Assignment 3, the students must choose which team member's project to enhance for the team project.

The teams follow an agile software development methodology resembling XP (including pair programming, unit testing, informal requirements, short iterations, and some other practices) due to the match between the project/team and the context in which XP works best (small, co-located, object-oriented programmers [27, 30]). Additionally, each student is assigned a team role, as is laid out in the TSP [28]:

- *Team leader*: leads the team and ensures that engineers report their progress data and complete their work as planned.
- *Development manager*: leads and guides the team in product design, development, and testing.
- *Planning manager*: supports and guides the team in planning and tracking its work.
- *Quality/process manager*: supports the team in defining the process needs and establishing and managing the quality plan.
- *Support manager*: supports the team in determining, obtaining, and managing the tools needed to meet its technology and administrative support needs.

Students are required to build a system according to the requirements (stated as user stories), thoroughly test the system, and create user documentation. Students must use at least one design pattern in their final project. The project covers the entire scope of the course thus far and also requires the students to assimilate some external technical knowledge on their own, manage time schedules, assign tasks, debug and troubleshoot. The teaching staff's role during the project is to answer requirements-related questions, to resolve technical issues, and to handle the problem of non-participatory students.

An important aspect of the project is the required weekly progress reports. At the end of the first week, the teams submit a Software Project Management Plan (SPMP). The SPMP lays out how their team will be organized, their configuration management/change tracking procedures, their choice of design pattern, and their preliminary schedule for completion. In the first week, the team also creates a team website upon which they are required to post a weekly Risk Analysis Form (RAF). Each week, on the RAF the students delineate each person's contribution for the prior week and the actual time spent for each contribution. They also lay out each person's responsibilities for the next week. Finally, they list the team's top ten risks for completing the project.

In lab each of the following weeks the students start out by having a Scrum/stand up meeting in which they all stand in a circle and say what they had accomplished in the past week, any problems they had, and what they will work on next. The teams also meet with their lab section's TA, demonstrate the requirements completed the prior week (and running the associated JUnit and FIT test cases), and discuss a set of requirements that they will implement in the coming week and that week's posted RAF. At the end of each weekly iteration, the teams are graded on how much they have accomplished on their chosen user stories. This encourages the students to begin work on the project early and to work consistently, rather than procrastinate.

While some students complain about the amount of work required on the project, many also say that it is the most enjoyable part of the course. The students' motivation is increased due to a project demonstration and competition at the end of the semester. In the final lab session, each team demonstrates their completed project to the class. Students get to see a myriad of solutions and user interfaces for the same requirements. The lab section votes on the best project. The teaching staff considers this popular vote in the choice of the best project for the laboratory. The teaching staff also reflects on how well the team worked together, the quality of the team documentation, and how well the team kept on schedule throughout the six weeks. The team who had the best project in the laboratory gets 5 points added to their final exam grade and continues to the finalist competition in lecture on the final day of class. The overall class winner is awarded 10 points added to their final exam grade as well as a small gift.

Ensuring collaborative participation

At the end of each paired homework assignment and twice during the project, students are required to evaluate their partners using the PairEval* system. Based upon the peer rating system by Kaufman et al. [31], the students are asked to choose one of the 13 key words in the bullets below (short descriptions are provided to the students as well) to describe the contribution of his or her partner:

1. *Excellent.* Consistently went above and beyond—tutored teammates, carried more than his/her fair share of the load
2. *Very good.* Consistently did what he/she was supposed to do, very well prepared and cooperative
3. *Satisfactory.* Usually did what he/she was supposed to do, acceptably prepared and cooperative
4. *Ordinary.* Often did what he/she was supposed to do, minimally prepared and cooperative
5. *Marginal.* Sometimes failed to show up or complete assignments, rarely prepared
6. *Deficient.* Often failed to show up or complete assignments, rarely prepared
7. *Unsatisfactory.* Consistently failed to show up or complete assignments, unprepared
8. *Superficial.* Practically no participation
9. *No show.* No participation at all

Students can also provide a textual explanation of the rating. If a student gives their partner a low overall rating (e.g. 'Marginal' or below), the partner is flagged and the teaching staff can review the evaluation more carefully. If the comments in the PairEval system suggest that one partner did not sufficiently participate in the homework assignment or project, then the professor will speak with the students involved to determine if any action needs to be taken. If, after investigation, the professor determines that a student made little or no effort on a partnered assignment, he or she will have his or her grade reduced accordingly and the partner's grade will be correspondingly increased. In our extensive experience with pairing students for homework assignments, prompt attention and severe consequence are essential for bringing potential 'freeloaders' back into contributing fairly to the success of the project. After the instructor handles a few of such instances, an environment of participation is created in the classroom and instances of freeloading become rare (though not nonexistent).

ENGAGING A DIVERSITY OF STUDENTS

Our approach to teaching software engineering is meant to give our students a diverse skill set to prepare them for their careers. As educators, we are concerned that our methods not only teach diverse skills, but also reach the diversity of students in our classroom. If you ask the average person in the United States what a computer science student looks like, that person will probably describe a nerdy, introspective, white male. A glance into the computer science classroom will undoubtedly produce such a specimen (or two or three), but the class will also have women, an assortment of ethnic minorities, and trendy, well-adjusted white males. Beyond the obvious surface differences, there are personality traits, learning styles, and career goals that are as varied among these students as the song lists on their MP3 players. What both the casual observer and the course instructor may not understand is that this plethora of students has vastly different learning needs and expectations.

We collected a variety of data to help paint a picture of the diversity of students in our classroom. The data collection process preserved individual student privacy rights and is FERPA compliant. Students' demographic information was obtained voluntarily, and participation in the

* http://agile.csc.ncsu.edu/pairlearning/paireval.php

study had no bearing on the students' grades in the software engineering course. The students who opted to participate in the study signed Institutional Review Board consent forms and could specify which data could be used in the study among gender, ethnicity, GPA, computer science GPA, and SAT scores. Participation was solicited by an investigator who was not a member of the teaching staff.

Demographics from the students in the undergraduate software engineering course at NCSU collected from the Fall 2004, Fall 2005, Fall 2006, and Spring 2007 semesters are shown in Table 1. There were a total of 253 students in these four semesters. Approximately half of the students at NCSU volunteered the gender and ethnicity information. Less than 8% of these students were non-Asian minorities, and only about 10% were women.

Collaboration and conscience—meeting the needs of the Millennial generation, women, and ethnic minorities

The emphasis on a socially relevant and practical project throughout the software engineering course is intentional. Sociologists have found that women feel compelled to find a means of serving others and work at this all their lives; doing so makes them comfortable and satisfied [32]. Other studies have found that ethnic minorities place similar importance on 'giving back' to their communities through socially-conscious work [33]. Furthermore, the U.S. Millennial generation as a whole tends to place special importance on social networks and interpersonal relationships and want to do 'something that matters' with their lives [34].

Collaboration on homework assignments and the class project, and active learning in the lab sessions are of paramount importance in our software engineering course. This emphasis is also grounded in students natural learning needs. Millennials' learning preferences tend toward teamwork and experiential learning, and their strengths include a collaborative style of working [34, 35]. They often learn better through discovery rather than by being told, and they prefer learning through participation rather than by learning by being told what to do [35]. Other studies have indicated that female students are concerned about the insularity of working alone for long periods of time, as they perceive to be the case with computer science and IT careers [36–38]. Furthermore, research has also shown that the success rate of under-represented minorities in science courses can be improved dramatically by shifting the learning paradigm from individual study to one that capitalizes on group processes, such as student work groups and student–student tutoring [39, 40].

Beneath the surface—diverse personality types and learning styles

The Myers–Briggs personality types [41] have served as a popular means of characterizing personality traits in both the classroom and the workplace. A considerable amount of work has been published on Myers–Briggs personality types in engineering education (e.g. [8, 9, 42]). Despite criticism for the unreliability of some personality tests, their misapplication, and lax experimental methods [43], the concepts behind personality types remain a valuable way of understanding the many dimensions upon which students can differ. The Myers–Briggs scale has four dimensions: Introvert–Extravert, Sensing–Intuition, Thinking–Feeling, and Judging–Perceiving (see the Appendix for a summary of each). Each dimension has different implications for teaching and learning. Similarly, the Felder–Silverman learning styles have been used to help students understand their own learning needs and to help professors better tailor their courses to different types of students [44]. The purpose of these learning styles is to help characterize the way in which students absorb and retain information. The Felder–Silverman scale has four dimensions: Active–Reflective, Sensing–Intuitive, Visual–Verbal, and Sequential–Global (see the Appendix for a summary of each).

Appealing to the many different types of personality types and learning styles can seem daunting. Traditional classrooms inherently seem to favor certain types over others, such as favoring

Table 1. Demographics of NCSU software engineering students

Millennial*		**Gender**		**Ethnicity**	
Yes	74.7%	Female	9.9%	African American	2.2%
No	25.3%	Male	90.1%	American Indian/Alaskan	2.2%
				Asian/Pacific Islander	6.5%
				Hispanic	2.9%
				White	86.2%
Total	253	**Total**	142	**Total**	139
Response rate†	100.0%	**Response rate**	56.1%	**Response rate**	54.9%

* Millennial students are those born in 1982 or later.
† Age is not protected information.

Intuitors, who prefer concepts, by speaking of general theories while neglecting the concrete data that helps Sensors. The structure of our software engineering course favors a more balanced approach in a number of ways. A balance of individual and collaborative work appeals to Introverts and Extraverts, and the hands-on lab sessions appeal to Active learners while periodic questions in lecture can be helpful to Reflective learners. Concepts are presented in lectures for Intuitors, and the implementation of these concepts is explored in labs for Sensors. For the Thinkers, who make decisions based on rationale and logic, we present course materials objectively and provide systematic comparisons of techniques, while for the subjectively-minded Feelers, we place special emphasis on the importance of collaboration and community. The orderly Judgers have structured syllabi and delineated expectations, while the flexible Perceivers are afforded the opportunity to adapt in a less rigid agile software development process. We provide written notes and lecture slides for the Verbal learners, supplemented by charts and diagrams for the Visual learners. In the classroom and labs, topics flow from one to the other (and are often revisited) for the benefit of the Sequential learners while, for the Global learners, we draw on outside disciplines for project inspiration and for problems analogous to those in the software domain.

All students in the undergraduate software engineering course at NCSU from 2004 to 2006 academic years took an online Myers–Briggs test* and the Index of Learning Styles Questionnaire† as part of their first homework assignment. The same is true of students in software engineering courses that followed our structure at North Carolina A&T‡ (NCAT) and Meredith College§ during Spring 2005 and Spring 2006. The results were recorded by the students in the PairEval system. The data were screened for possible falsified entries prior to analysis by examining for students' Myers–Briggs scores for entries that did not match the discrete numerical values possible for this test. After the screening, we had 396 and 405 Myers–Briggs and Learning Styles responses respectively. Some students who completed the Index of Learning Styles Questionnaire did not complete the Myers–Briggs test. The Myers–Briggs and Learning Styles responses are shown in Table 2. The varied approach to the software engineering course helps us to respond to the varied needs of our diverse set of students.

Student reaction and course impact

The student response to the course has been favorable. We conducted interviews with students taking the course and reviewed course retrospectives to gain insight on the student-perceived benefits of the course and to understand how we can further improve the course. For more student responses to the course, particularly those of women and ethnic minorities, please see [46–48]. The importance of real-world, practical projects was appreciated by the students.

> A lot of projects done in school seem to miss on usefulness. However, right from the go it was clear the usefulness and importance of our project. [48]

> I liked how it was something that would actually be used by people and not some moronic assignment that calculates deer population that has no relevance to the real world.

Collaboration with peers and a lab environment that enabled the students to work as teams on a common project was also appreciated.

> You can share each other's knowledge like sometimes it's better to learn from your peers than from the instructor because they might know how to relay it better. So when you're working in groups you're also learning. [46]

> I went for an interview with this company . . . he showed me the [Extreme Programming] area, where they do pair programming and it looked just like where we had the lab. That amazed me . . .I saw people programming together and right there I just wanted to work for the company, I wanted to quit school and start working with them.

While we were not enthusiastic of the 'quitting school' part, this student quote captures the importance of collaboration and what a strong motivator and enabler it can be to student interest and learning.

Statistical analysis of class performance

The emphasis on a balanced of agile and plan-driven techniques, individual and group work, hands-on and hands-off learning also brought a balance to the student's performance in the course. Previous studies have suggested that students with particular personality types and learning styles, typically Sensors, Perceivers, Active learners, and Verbal learners, do not perform as well as their peers in engineering courses [5–9].

We compared the means of the students' total class grades in all MBTI and LS dimensions (e.g., Extravert vs. Introvert, Visual vs. Verbal). To assure that we could legitimately collate the data from all semesters, we first performed chi-square tests, which yielded no statistically-significant differences at the $p < 0.05$ level in the distributions of the dimensions across all semesters. To ensure that we used the appropriate statistical test for comparing means, all MBTI and LS dimensions were assessed for non-normality using the Shapiro–Wilk test, and only the Sensing–Intuitive dimension on the LS scale had a non-normal class score distribution. We then used *t*-tests for all normally-distributed populations, and the non-

* http://www.humanmetrics.com/cgi-win/JTypes2.asp. This test has not been evaluated for reliability.

† http://www.engr.ncsu.edu/learningstyles/ilsweb.html. A thorough reliability evaluation may be found in [45].

‡ http://www.ncat.edu

§ http://www.meredith.edu

Table 2. Personality type and learning style categorical breakdown

Myers–Briggs type indicators*				**Felder–Silverman learning styles**			
Category		**Type**		**Category**		**Type**	
E—Extraversion	45.5%	ENFJ	9.9%	A—Active	44.4%	ANBG	0.5%
I—Introversion	54.5%	ENFP	1.3%	R—Reflective	55.6%	ANBQ	1.7%
		ENTJ	16.9%			ANVG	8.1%
S—Sensing†	34.9%	ENTP	1.8%	S—Sensing†	57.8%	ANVQ	6.4%
N—Intuition†	65.1%	ESFJ	4.0%	N—Intuition†	42.2%	ASBG	0.5%
		ESFP	1.0%			ASBQ	1.5%
T—Thinking	72.7%	ESTJ	9.3%	V—Visual	80.5%	ASVG	7.7%
F—Feeling	27.3%	ESTP	1.3%	B—Verbal	19.5%	ASVQ	18.0%
		INFJ	4.3%			RNBG	3.0%
J—Judging	81.8%	INFP	1.0%	G—Global	39.0%	RNBQ	2.5%
P—Perceiving	18.2%	INTJ	21.7%	Q—Sequential	61.0%	RNVG	10.4%
		INTP	8.3%			RNVQ	9.6%
Sample size	396	ISFJ	4.5%	**Sample size**	405	RSBG	3.0%
		ISFP	1.3%			RSBQ	6.9%
		ISTJ	11.1%			RSVG	5.9%
		ISTP	2.3%			RSVQ	14.3%

* Accuracy of the estimates of this test is not available. The Felder–Silverman ILS questionnaire was formally assessed for reliability [45].

† Differences in the distribution of the Sensing–Intuition dimension are the likely result of construct differences between the typology and ILS questionnaires.

parametric Wilcoxon–Mann–Whitney test for the non-normal population to test for differences in means. *No statistically significant differences in the mean total score were found along any dimension at the $p < 0.05$ level.*

We also looked for any difference in the performance of Millennials vs. non-Millennials. Interestingly, there *was* a statistically significant difference in the performance of Millennials and non-Millennials—the Millennials as a whole performed better by an average of 3.5%. This finding bears further investigation as we desire for all students groups to perform equally well. We again conducted tests to ensure the reliability of the analysis. We found that the Fall 2004 semester had statistically significantly more non-Millennials than the other semesters (chi-square). We also found that the Fall 2005 semester generally received a lower overall grade (ANOVA). The net effect of these differences is to *increase* the mean non-Millennial score in our data. Since a statistically significant difference in the mean scores of Millennials and non-Millennials exists despite this increase, we assume our tests are valid. Unfortunately, the small sample size of women and ethnic minorities was too small and varied for an analysis of the distribution to be informative.

CONCLUSIONS

For the last seven years, we have evolved a core undergraduate software engineering class that prepares students for the demands of industry by inculcating them with sound software engineering practices that bridge both traditional plan-driven and newer agile practices. We have presented the content layout of the course, the progression of topics, and the facilities used in our course. We have also discussed the diverse learning needs of today's Millennial students, women, ethnic minorities and a variety of personality types and learning styles. Whereas previous studies had shown systematic differences in the performance of different personality types and learning styles [5–9], our analysis suggests that all types perform equally well in this course. Women, minorities, and under-represented personality types and learning styles have commented on the success of this course at addressing their specific needs [41–43]. The spectrum of students in our course reinforces the need for a new pedagogy conscious of this diversity. We recognize that there are some groups that we have not specifically addressed in this paper, such as foreign-language students, students with disabilities, and continuing education students.

Through collaboration, hands-on learning, and a focus on practicality, we have striven to create an environment that is effective at engaging students and fostering further interest in the computer science discipline. We have also created a course whose content provides the students with a wide variety of tools and techniques to prepare them better for the diverse needs of an evolving software industry. We hope that educators will expand and improve upon the techniques and ideas used in our software engineering class for other computer science courses.

SUMMARY

The software industry uses a mixture of plan-driven and agile techniques, and educators must prepare students for industry needs while creating an effective educational environment that appeals to a diverse student population. We describe the undergraduate course in software engineering at

North Carolina State University, which teaches both agile and plan-driven practices while emphasizing collaborative and active learning. We present demographics, personality types, and learning styles from 400 students, and provide statistical analyses and student testimonials on the impact of our course. Students have reacted favorably to the course and are better prepared to meet the diverse needs of industry.

Acknowledgements—We would like to thank Barry Koster of Meredith College and the late Sung Yoon of North Carolina A&T for their invaluable contributions to this study, as well as Jason Osborne for his helpful advice. This material is based upon the work supported by the National Science Foundation under Grants ITWF 00305917 and BPC 0540523. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

1. B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley, Boston, MA, (2003).
2. A. Cockburn, *Agile Software Development*, Addison Wesley Longman, Reading, MA, (2001).
3. J. Sargent, An overview of past and projected employment changes in the professional it occupations, *Computing Research News*, **16**(3), 2004, pp. 1–21.
4. Computing Research Association, 2005–2006 Taulbee survey, *Computing Research News*, **19**(3), 2007, pp. 7–22.
5. L. Thomas, M. Ratcliffe, J. Woodbury and E. Jarman, Learning Styles and Performance in the Introductory Programming Sequence, SIGCSE '02, (2002) pp. 33–37.
6. J. Allert, Learning Style and Factors Contributing to Success in an Introductory Computer Science Course, IEEE International Conference on Advanced Learning Technologies (ICALT '04), (2004) pp. 385–389.
7. R. M. Felder, G. N. Felder and E. J. Dietz, The effects of personality type on engineering student performance and attitudes, *Journal of Engineering Education*, **91**(1), 2002, pp. 3–17.
8. E. S. Godleski, Learning Style Compatibility of Engineering Students and Faculty, *Frontiers in Education (FIE '84)*, (1984) pp. 362–364.
9. M. H. McCaulley, The MBTI and individual pathways in engineering design, *Journal of Engineering Education*, 80, 1990, pp. 537–542.
10. B. Boehm, Get ready for agile methods, with care, *IEEE Computer*, **35**(1), 2002, 64–69.
11. W. W. Royce, *Managing the Development of Large Software Systems: Concepts and Techniques*, IEEE WESTCON, Ch. 3 (1970).
12. B. Boehm, A spiral model for software development and enhancement, *Computer*, **21**(5), 1988, pp. 61–72.
13. J. Highsmith, *Agile Software Development Ecosystems*, Addison-Wesley, Boston, MA (2002).
14. V. R. Basili and A. J. Turner, Iterative Enhancement: A practical technique for software development, *IEEE Transactions on Software Engineering*, **1**(4), 1975, pp. 266–270.
15. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, MA, (2000).
16. K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*, Prentice-Hall, Upper Saddle River, NJ, (2002).
17. S. R. Palmer and J. M. Felsing, *A Practical Guide to Feature-Driven Development*, Prentice Hall PTR, Upper Saddle River, NJ, (2002).
18. S. Ambler, Survey says . . . agile has crossed the chasm, *Dr. Dobb's Journal*, **32**(7), 2007.
19. L. Williams and R. Kessler, *Pair Programming Illuminated*, Addison Wesley, Reading, MA, (2003).
20. M. Rappa, S. E. Smith, A. Yacoub and L. Williams, OpenSeminar: Web-based collaboration tool of open educational resources, *International Conference on Collaborative Computing (CollaborateCom '05)*, (2005).
21. L. Layman, L. Williams and K. Slaten, Note to self: make assignments meaningful, *ACM Technical Symposium on Computer Science Education (SIGCSE '07)*, (2007) pp. 459–463.
22. M. Howard and S. Lipner, *The Security Development Lifecycle*, Microsoft Press, Redmond, WA, (2006).
23. G. McGraw, *Software Security: Building Security In*, Addison-Wesley, Upper Saddle River, NJ, (2006).
24. C. Kaner, J. Bach and B. Pettichord, *Testing Computer Software*, Wiley, New York, (1999).
25. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, (1995).
26. I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Wokingham, UK, (1992).
27. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, MA, (2005).
28. W. S. Humphrey, *Introduction to the Team Software Process*, Addison Wesley, Reading, MA, (2000).
29. B. Boehm and R. Turner, Using risk to balance agile and plan-driven methods, *IEEE Computer*, **36**(6), 2003, pp. 57–66.
30. R. Jeffries, A. Anderson and C. Hendrickson, *Extreme Programming Installed*, Addison Wesley, Upper Saddle River, NJ, (2001).
31. D. B. Kaufman, R. M. Felder and H. Fuller, *Peer Ratings in Cooperative Learning Teams*, American Society for Engineering Education, (1999).
32. J. Margolis and A. Fisher, *Unlocking the Clubhouse: Women in Computing*, The MIT Press, Cambridge, MA, (2002).

33. S. Shue, J. L. Vest and J. Villarreal, *Philanthropy in Communities of Color*, Indiana University Press, Bloomington, IN, (1999).
34. D. Oblinger, Boomers, Gen-Xers, and Millennials: Understanding the New Students, Educause Review, **38**(4), 2003, pp. 37–47.
35. D. Oblinger and J. Oblinger, Is It age or IT: First steps toward understanding the net generation, in *Educating the Net Generation* (D. G. Oblinger and J. L. Oblinger, Eds.), Educause, (2005).
36. J. Margolis and A. Fisher, Geek mythology and attracting undergraduate women to computer science, *Joint National Conference in Engineering Program Advocates Network and the National Association of Minority Engineering Program Administrators*, (1997).
37. American Association of University Women Education Foundation, *Educating Girls in the New Computer Age*, http://www.aauw.org/member_center/publications/TechSavvy/TechSavvy.pdf, (2000).
38. P. Freeman and W. Aspray, *The Supply of Information Technology Workers in the United States*, http://www.cra.org/reports/wits/cra.wits.html, May 31, (1999) (viewed 2007).
39. C. E. Nelson, Student diversity requires different approaches to college teaching, even in math and science, *American Behavioral Scientist*, **40**(2), 1996, pp. 165–175.
40. U. Treisman, Studying students studying calculus: A look at the lives of minority mathematics students in college., *The College Mathematics Journal*, **23**(5), 1992, pp. 362–372.
41. G. Lawrence, *People Types and Tiger Stripes*, Center for Applications of Psychological Types, Gainesville, FL, (1994).
42. A. Thomas, M. R. Benne, M. J. Marr, E. W. Thomas and R. M. Hume, The Evidence remains stable: The MBTI predicts attraction and attrition in an engineering program, *Journal of Psychological Type*, **55**, 2000, pp. 35–42.
43. S. McDonald and H. M. Edwards, Who should test whom?, *Communications of the ACM*, **50**(1), 2007, pp. 66–71.
44. R. M. Felder and L. K. Silverman, Learning and teaching styles in engineering education, *Journal of Engineering Education*, **78**(7), 1988, pp. 674–681.
45. R. M. Felder and J. Spurlin, Applications, reliability and validity of the index of learning styles, *International Journal of Engineering Education*, **21**(1), 2005, pp. 103–112.
46. L. Williams, L. Layman, K. M. Slaten, C. Seaman and S. B. Berenson, On the impact of a collaborative pedagogy on african–american millennial students in software engineering, *International Conference on Software Engineering (ICSE '07)*, electronic proceedings (2007).
47. K. Slaten, M. Droujkova, S. Berenson, L. Williams and L. Layman, Understanding student perceptions of pair programming and agile software development methodologies: Verifying a model of social interaction, *Agile 2005*, 2005, pp. 323–330.
48. L. Layman, T. Cornwell and L. Williams, Personality types, learning styles, and an agile approach to software engineering education, *ACM Technical Symposium on Computer Science Education (SIGCSE '06)*, (2006) pp. 428–432.

APPENDIX

The Myers–Briggs scale has four dimensions:

- *Introvert–Extravert*. Introverts are generally introspective and are energized by spending time alone, whereas extraverts thrive in a group setting.
- *Sensing–Intuition.* Sensors prefer information gathered through experience and are attentive to details, while intuitors prefer abstract concepts and are bored by details, preferring innovative thoughts instead.
- *Thinking–Feeling.* Thinkers rely on objective rationalization to make decisions and are considered to be impartial, whereas feelers are more likely to make subjective decisions based on social considerations rather than strict logic.
- *Judging–Perceiving*. Judgers are typically orderly people who prefer rigid structure and planning but may ignore facts that do not fit their plan or structure, whereas perceivers do little planning and work spontaneously but are more open to facts that do not conform to their views.

The Felder–Silverman scale has four dimensions:

Active–Reflective. Active learners learn best by experimentation and working with others, while reflective learners learn more by thinking things out on their own.

Sensing–Intuitive. The sensing–intuitive dimension is intended to be the same as in the Myers–Briggs scale.

Visual–Verbal. Visual learners absorb information best through pictures, graphs, and charts, whereas verbal learners prefer written or spoken explanations.

Sequential–Global. Sequential students learn in orderly, incremental steps with one point or fact connecting to the next, whereas global learners have trouble learning fact-by-fact and learn in cognitive leaps after accumulating all the facts.

Lucas Layman is a Ph.D. candidate in the Department of Computer Science at North Carolina State University. He earned his B.S. in computer Science from Loyola College and his M.S. from N.C. State University. His research has focused on empirical software

engineering and agile software development techniques, the psychology of computer programming, and computer science education issues specifically relating to women and ethnic minorities. He recently interned at Microsoft Research studying collaboration in software development, software reliability, and software process management.

Laurie Williams is an Associate Professor at North Carolina State University. She received her undergraduate degree in Industrial Engineering from Lehigh University. She also received her MBA from Duke University and her Ph.D. in Computer Science from the University of Utah. Prior to returning to academia to obtain her Ph.D., she worked in industry, for IBM, for nine years. Dr. Williams is the lead author of Pair Programming Illuminated and a co-editor of *Extreme Programming Perspectives*. Dr. Williams has done several empirical studies on Extreme Programming and its development practices, pair programming and test-driven development.

Sally Berenson is the Yopp Distinguished Professor of Mathematics Education in the Department of Curriculum and Instruction at the University of North Carolina Greensboro and directs the Center for Research in Mathematics and Science Education. Dr. Berenson's current research has focused on examining young women's persistence and career interest in mathematics and information technology. In addition to these gender studies, Dr. Berenson pursues studies in mathematics teacher preparation working to integrate content and pedagogy. She has published numerous articles on Lesson Plan Study and the application of this research tool to undergraduate instruction in order to build a model of how prospective math teachers learn what and how to teach.

Kelli M. Slaten is an Assistant Professor of Mathematics Education in the Mathematics and Statistics Department of the University of North Carolina Wilmington. She is also the Director of Secondary Mathematics Education. Her research interests include the teaching and learning of mathematics through the use of multiple representations and the use of technology, gender and cultural issues in education, and qualitative research methods. She received her Ph.D. in Mathematics Education from North Carolina State University in 2006.

Mladen A. Vouk received his Ph.D. from King's College, University of London, UK. He is Department Head and Professor of Computer Science, and Associate Vice Provost for Information Technology at N.C. State University, Raleigh, NC, U.S.A. He has extensive experience in both commercial software production and academic computing. He is the author/co-author of over 200 publications. His research and development interests include software reliability engineering, scientific workflows and computing, information technology assisted education, and high-performance virtualized computing systems and networks. He is an IEEE Fellow, and recipient of the IFIP Silver Core award.