Personality Types, Learning Styles, and an Agile Approach to Software Engineering Education

Lucas Layman Travis Cornwell Laurie Williams

North Carolina State University, Department of Computer Science
Campus Box 8207, Raleigh, NC 27695
+1-919-513-5082
[lmlayma2, twcornwe]@ncsu.edu, williams@csc.ncsu.edu

ABSTRACT

This paper describes an initiative at North Carolina State University in which the undergraduate software engineering class was restructured in layout and in presentation. The change was made from a lecture-based coursed that followed the waterfall method to a lab-oriented course emphasizing practical tools and agile processes. We examine the new course layout from the perspective of Myers-Briggs personality types and Felder-Silverman learning styles to discuss how the new software engineering class format appeals to a wide variety of students. The new course format resulted in some of the highest student evaluations in recent course history. It is now the standard for the undergraduate software engineering course at the university and has since been used in other North Carolina institutions.

Categories and Subject Descriptors

K.3.2. [Computing Milieux]: Computing and Information Sciences Education – *computer science education*

General Terms

Human Factors

Keywords

Software engineering education; agile methods; learning styles; personality types

1. INTRODUCTION

Historically, the software engineering course at North Carolina State University (NCSU) was loosely based on the waterfall software development process. The course had three 50-minute lectures each week, and a final group project that employed the waterfall process model. Unfortunately, student evaluation markings for this course were consistently among the lowest in the department. Students complained vociferously about both the course content and the course presentation. In 2003, NCSU instituted a new approach to teaching undergraduate software engineering that used agile software development processes and focused on tools and techniques rather than lecture-based concepts. This new approach also involved a weekly lab component that took the place of the third lecture. The new approach was well-received by students, and the student evaluation markings were among the highest for the course in recent history. In this paper, we describe the software engineering course that has now become standard at NCSU. We discuss the basic course layout and teaching principles used in both the lecture sections and in the lab sessions.

To understand the appeal of the new course structure, we examine our class from the standpoint of Myers-Briggs personality types [5] and Felder-Silverman learning styles [3]. We provide the personality type and learning style distributions of the students in the Fall 2004 undergraduate software engineering class. Each dimension of the Myers-Briggs and Felder-Silverman scale requires specific consideration when creating a teaching style that appeals to as many students as possible. This paper discusses how a hands-on, collaborative, agile-based approach to teaching software engineering seems to appeal to a wide variety of student personality types and learning needs. We supplement our explanations with student testimony gathered at the end of the course.

The remainder of the paper is organized as follows: Section 2 discusses related work, and Section 3 presents the course layout. Sections 4 and 5, respectively, provide a discussion of how this software engineering course appeals to different Myers-Briggs personality types and Felder-Silverman learning styles profiles. We conclude in Section 6.

2. RELATED WORK

This section provides a brief introduction to the Myers-Briggs personality types and Felder-Silverman learning styles. We also summarize some related empirical studies.

2.1 Myers-Briggs

The Myers-Briggs personality types [5] 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 (e.g. [4, 6, 7]). 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

SIGCSE'06, March 1–5, 2006, Houston, Texas, USA.

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.

Myers-Briggs personality may affect student performance in the engineering classroom. Felder found that, in terms of course grades, introverts outperformed extraverts, intuitors typically outperformed sensors (except in hands-on, "real-world" classes), thinkers outperformed feelers, and judgers typically outperformed perceivers. These findings are similar to those found in [4, 6].

2.2 Learning styles

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 [3]. 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. 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.

Some work has been done on the learning styles of computer science students. Thomas, et al. found that reflective learners typically outperform active learners and verbal learners outperform visual learners with respect to exam grades and course grades [8]. Similar course performance records were found by Allert [1]. [2]

3. COURSE LAYOUT

The overall goal of the software engineering course at NCSU is to teach students practical techniques and tools that they will encounter in professional software development. As with many software engineering courses, the lecture sessions often center on concepts and theories, such as discussions of software processes and testing strategies. However, the weekly lab sessions are the focus of the course wherein the students receive hands-on experience with the concepts they have been taught in the classroom. In the lab sessions, students participate in project planning, learn to use components of the Eclipse IDE[1], become familiar with testing tools such as JUnit[2], write requirements documents, and so forth.

During the first nine weeks of the class, the students are given four homework assignments of one-three weeks each. The first assignment is to create a personal webpage so that other students may familiarize themselves with their peers, become familiar with their schedules, etc. This is important as the students will be working with each other throughout the semester. Two of the homework assignments are done in pairs, and generally focus on understanding and applying a design pattern and a new technique (such as unit testing or version control) to a problem. The remaining homework assignment is done alone and follows the same layout as the paired homework assignments, but also affords the student the opportunity to compare working alone to working with a partner. The course also contains a midterm and a final exam, which are done individually, and test the students' comprehensive knowledge of the course material

During the last six weeks of the semester, the lab sessions are reserved for the student teams to work on their group projects, wherein they must apply all of the tools they have learned previously. 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. Students are required to build a system according to the specification, thoroughly test the system, and create user documentation. The projects require 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. The final project is non-trivial, and is meant to cover the entire scope of the course thus far and requires a significant number of person-hours to complete. Students often complain about the amount of work required on the project, but also say that it is the most enjoyable part of the course. One of the most important aspects of the project is the required weekly progress. At the end of each weekly iteration, the student teams meet with their lab section's TA and select a set of user stories they will implement in the coming week. The students are graded on how much they accomplished on their chosen user stories at the end of the weekly iteration. This encourages the students to begin work on the project early and to work consistently, rather than leaving all of the work until the end.

A final aspect of student grading are peer evaluations. At the end of each paired homework assignment and twice during the project, students are required to evaluate their partners using the PairEval[3] system. If a student gives an overall rating of their partner of five or less, 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 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 their grade reduced accordingly.

[1] www.eclipse.org

[2] www.junit.org

[3] http://agile.csc.ncsu.edu/wiki/doku.php?id=tools#paireval

4. PERSONALITY TYPES

Instructors must expect to encounter all personality types in their classrooms. Appealing to each personality type does not necessarily require a complete rewrite of an instructional approach, but can help make the class more engaging and memorable (in a good way) for the students. The Myers-Briggs personality types for the students in the Fall 2004 NCSU software engineering class are shown in Table 1. The distributions in Table 1 are typical of other findings of engineering students in general and computer science students in particular [2, 4]. One exception is the sensing-intuition dimension, in which our class had a high proportion of intuitors whereas other studies have shown a greater proportion of sensors. We discuss each of the Myers-Briggs dimensions in turn and how our teaching approach aligns with the needs of all personality types.

Table 1 – MBTI categorical breakdown

Myers-Briggs Type	Abbreviation	N	%
Extraversion	E	31	46.27%
Introversion	I	36	53.73%
Sensing	S	12	17.91%
Intuition	N	55	82.09%
Thinking	T	50	74.63%
Feeling	F	17	25.37%
Judging	J	51	76.12%
Perceiving	P	16	23.88%

4.1 Extravert-introvert personalities

Contrary to popular views, there were near equal numbers of extraverts and introverts in our class. This has several implications on teaching style. The true defining feature of extraverts and introverts is how they motivate and recharge themselves: extraverts find their energy from working in groups, and introverts find their energy from working alone. A balance of extraverts and introverts in the classroom motivates a balance of individual and group work.

Most courses inherently favor introverts because of the focus on individual study and performance. In our software engineering course, students are assigned weekly readings from the course textbook, and one of the homework assignments is done alone. The midterm and final exams are individual efforts and account for 45% of the final grade.

We also place a strong emphasis on collaboration in our course. Two of the three homework assignments are done in pairs, and the group project is a team effort. Thus, almost 40% of a student's total grade is dependent on group work, a proportion almost equal to that of the individual assignment and exams. The majority of the students in the class, including introverts, enjoyed the emphasis on pair programming. One student commented, "*My favorite part about pair programming is that many times when working solo, I get stuck on the logics of my program, and having a partner often avoids this.*" Another student observed, "*Our team members seemed to communicate better [in pairs] and it was easier to make design decisions when you had another person with you at all times.*"

4.2 Sensing-intuition

The sensing-intuition dimensions of both the Myers-Briggs personality types and Felder-Silverman learning styles concern how people prefer to receive data. Most instructors teach in a style that suits intuitors [3] by using lectures and presentations to emphasize concepts, as opposed to factual data. In our course, the lectures convey ideas such as testing strategies, process models, and quality assurance approaches. The oral lectures in our class are supplemented by presentation slides or text written on a whiteboard.

Sensors can become bored with concept-oriented lectures; they need to see facts and real world application. An agile process approach offers an appealing element to sensors: rapid feedback. Most agile processes are constructed to respond to changes in requirements, changes in personnel, and changes in technology. To facilitate change, it is important to have constant feedback on the development process to assess the project's current status. This feedback can take many forms, including daily meetings with developers, constant test results, and more.

In our class, consistent feedback was facilitated through several means. Mandatory unit testing and automated acceptance testing of all assignments and the project was required. These tools allowed students to see, in graphical representations on the computer screen, measurements of their testing progress and thoroughness. One sensing student pointed out, "*[JUnit and acceptance testing were] useful for when we made changes to the code because we would change a small piece, run the test cases, if they all passed then we most likely didn't break anything.*" Another sensing student noted how the acceptance testing "*helped to visualize what the output was supposed to look like.*" Almost all of the sensing students commented in their retrospectives on the benefits of testing incrementally throughout the project.

During the semester project, another form of feedback provided in the weekly iteration meetings during the lab sessions. In these meetings, the student teams filled out a detailed form outlining their progress and received feedback from their TA. One sensing student observed, "*[Weekly meetings] helped to keep everyone on pace each week with what needed to be accomplished. This was probably the most helpful thing, as our group had no last minute crunch to complete work.*" Similarly, another sensing student stated, "*I think the feedback from each iteration was helpful in deciding if we were on the right track.*" Every sensing student in the class who turned in a retrospective noted the benefits of these meetings either to assess their progress or to obtain feedback.

4.3 Thinking-feeling

Thinkers are rational and logical in their decision making processes, feelers make decisions based on intuition and personal consideration. At first glance, most engineering classrooms would seem to require a thinker. Most course materials are presented objectively as matters of fact, e.g. "This is how you write a test case" or "These are the steps of the waterfall process." In many ways, the procedures, facts, and problem solving strategies of engineering disciplines are ideal for thinkers. In our course, much of the material is presented in a way that caters to thinkers. Lectures focus on strategies for solving common problems (e.g., "these are the steps you take to create a class diagram"), the rationale behind certain tools and techniques is discussed (e.g., "code inspections yield a 60% reduction in maintenance cost"), and exams reward critical thinking and application of learned techniques.

Appealing to feelers is more of a challenge. Personal consideration and emphasis on human elements and social relevance are particularly important to feelers. It is important that the work they do reaches beyond a simple exercise and will have impact beyond determining what grade they receive. For the final project in our software engineering course, we strive to present the students with a problem that is practical and relevant to them as computer science students. For this particular class, the students were tasked to write a bug tracking system that was integrated into the Eclipse IDE and made use of a remote database. The Eclipse IDE is a commercially popular tool and several of the students had used it in various jobs and internships. Bug tracking is a part of every day life for software developers. One feeling student commented, "*A lot of projects done in school seem to miss on usefulness. However, right from the get go it was clear the usefulness and importance of our project. I am so satisfied with the out come of the project, that if I was working on something with other people, I would use this [system]*." Another student noted, "*I enjoyed the project mainly because I like programming, but also because I like the idea of working on something so practical and 'real-world' as an Eclipse plug-in.*"

4.4 Judging-perceiving

In our class, as is typical with most engineering classrooms, judgers hold a majority over perceivers. Judgers tend to be organized, decisive, and they like concise, concrete explanations. On the other hand, perceivers are flexible, open to change, and are comfortable with ambiguity.

A well-organized and clearly presented lecture will often contain the type of information that appeals to judgers. In our class, we strive to order lectures in a rational manner, with one thought flowing easily to the next. The course textbook and presentation slides are organized in assorted lists and tables for clarity. Exam questions are as concise as possible, and several iterations of an exam are passed around the teaching staff to disambiguate the text as much as possible. Similar efforts are made in the specifications for the homework assignments and the group project. Though, with the group project in particular, there are inevitably some clarifications that need to be made to the system requirements. In general, we strive to make the course as concise and as orderly as possible.

Appealing to perceivers is more challenging. Course structure necessitates an orderly, planned syllabus, due dates for homework assignments, and exam times; the class structure itself is inflexible. Agile processes can offer to perceivers some measure of flexibility. In our class, the six week group assignment is managed on a weekly basis. That is, students evaluate their work at the end of each week and adjust their schedules accordingly.

The features the students are asked to implement in the group project are presented as "user stories," a type of requirement. The user stories are designed to be as independent as possible of each other. This minimizes the need for the students to wait on other functionality to be developed before working on certain items. By designing for independent user stories and evaluating student progress on a weekly basis, the student groups can assess the work that they accomplished in the previous week and make adjustments as necessary. Some groups abandoned current work items because they were progressing too slowly, or because the work was not essential to the finished product. As one perceiving student noted, "*Feedback from the lab TA and, more importantly, team members at the end of each iteration was most valuable in making the necessary time and procedural adjustments to better estimate the amount of work that could actually be completed in future iterations.*" This measure of flexibility and control offered over the development process can help appeal to the flexibility-minded perceivers.

5. LEARNING STYLES

The Felder-Silverman learning styles are a way of characterizing the ways that students absorb and process information. Felder contends that a misalignment between these learning styles and the teaching styles of professors are a problem in the classroom [3]. We discuss each of the learning style dimensions with the exception of the sensing-intuitive dimension. According to Felder, this sensing-intuitive dimension is taken from the Myers-Briggs sensing-intuition dimension and thus we omit it from this section. The learning style categorical distributions for the students in our class are presented in Table 2.

Table 2 – LS categorical breakdown

Learning Style	Abbreviation	N	Percentage
Active	A	29	43.28%
Reflective	R	38	56.72%
Sensing	S	38	56.72%
Intuitive	U	29	43.28%
Visual	V	56	83.58%
Verbal	B	11	16.42%
Sequential	Q	36	53.73%
Global	G	31	46.27%

5.1 Active-reflective

The active-reflective dimension concerns how students process information. Active learners form thoughts through active experimentation and application, whereas reflective learners digest information through introspection and careful thought.

While most lecture-oriented classes may, on the surface, appeal to the reflective learner, this is not always the case. Like many lectures-based courses, our software engineering course contains weekly reading assignments and presentation slides for the students to review and reflect on their own. However, for the lectures themselves to be effective, it is necessary to allow the students to reflect on the material during the lecture. In our class lectures, there is a break every 10-15 minutes in which the students are given a small task wherein they must think about the material that has just been presented to them. These tasks may simply be to come up with a question about the material or to complete a short exercise. The students may work on their own during these small tasks or discuss it with students sitting nearby. This short break keeps the students engaged and active throughout the lecture, as opposed to a block 50-minute lecture that may leave students disinterested or, worse, asleep at the end.

The periodic exercises in the lectures are also beneficial to the active learners, who are best able to process their knowledge when experimenting or applying what they have learned. The appeal to the active learner in our software engineering class lies in the weekly lab sessions. In these lab sessions, the students receive training with practical tools that correspond with the class lectures. For example, students learn testing strategies in the lecture one week early in the semester. That same week, the

students learn how to use the JUnit testing framework in lab and apply it to a problem as part of their homework assignments. This method appeals to students because of the hands-on experimentation and practical relevance of their lab sessions.

5.2 Visual-verbal

The visual and verbal dimension deals with the channels through which students perceive information. Visual learners prefer charts, symbols, pictures, and drawings, whereas verbal learners prefer written text or speech. Most students are visual learners, yet most lectures are presented as text slides accompanied by an instructor's lecture. The appeal to visual and verbal learners is largely dependent on presentation and not course framework.

We include pictures and charts in lecture materials when possible and relevant. For homework assignments and for the group project, we provide use case diagrams to detail the requirements specification (when appropriate). Furthermore, students are also required to draw class diagrams and sequence diagrams for their homework assignments and for the group project to help supplement their understanding of the system.

5.3 Sequential-global

The structure of our software engineering course inherently appeals to the sequential learner, as does the structure of most engineering courses. Concepts and tools are taught in a logical progression from one to the next, allowing the students to make cognitive ties between new material and the material they have just learned. For example, students in the class learn about traditional requirements specifications, followed by use cases, followed by a use case lab, followed by agile requirements practices, followed by project management, and so forth.

Appealing to global-minded students is more challenging and is based prominently on the communication skills of the teaching staff. Global learners need to understand how the current material relates to their past experiences and prior knowledge so that they can make large cognitive leaps in understanding. Analogy is a useful technique to help communicate with global learners. When speaking about a new technique, concept, or tool, we try to relate it to something outside of the software engineering classroom. For example, the act of validating requirements can be likened to a car inspection, wherein a certain set of criteria has to be met before the car is legal to drive. Effective comparisons are often hard to come by, but are very useful in reaching global learners. Not only will a useful analogy help students to better understand new material, it can help them to retain the knowledge by aligning it with a familiar subject.

6. CONCLUSION

We have discussed our course structure and teaching approach to an undergraduate software engineering class at NCSU. The combination of lecture and lab work, the use of an agile process model, and an awareness of the learning needs of different types of students has helped us to create a successful learning environment. We believe that we were able to create a better learning environment for students by instituting a course approach that appeals to a wide variety of personality types and learning styles. We have presented our rationale behind this claim and supplemented it with student testimony. This particular class marked the second time this pedagogy was used in the software engineering course at NCSU. The class received some of the highest student evaluation marks in the recent history of the course (it should be noted that most of the students also complained of how much work the class entailed). The student evaluation ranking also put the software engineering class well above the department average. The approach to the software engineering class in this paper is now the standard pedagogy for all NCSU software engineering classes and has been used at two additional North Carolina universities.

We are currently investigating the relationships between personality type, learning styles, and student grades in the software engineering class. We are also investigating cases where students consistently received poor grades in the course to determine what their characteristics are and what can be done to help them. This research is part of a larger study that aims to improve the interest and retention of women and minorities in the IT field by better understanding the learning needs and confounding factors that cause these students to avoid or leave the field. It is our hope that a better understanding of the learning needs of these students will lead to a more effective and engaging classroom.

7. ACKNOWLEDGEMENTS

This material is based upon the work supported by the National Science Foundation under the Grant No. 00305917. 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.

8. REFERENCES

[1] J. Allert, "Learning Style and Factors Contributing to Success in an Introductory Computer Science Course," proceedings of IEEE International Conference on Advanced Learning Technologies (ICALT '04), Joensuu, Finland, 2004, pp. 385-389.

[2] L. F. Capretz, "Personality Types in Software Engineering," *International Journal of Human-Computer Studies*, vol. 58, 2003, pp. 207-214.

[3] R. M. Felder and L. K. Silverman, "Learning and Teaching Styles in Engineering Education," *Engineering Education*, vol. 78, 1988, pp. 674-681.

[4] E. S. Godleski, "Learning Style Compatibility of Engineering Students and Faculty," proceedings of Frontiers in Education (FIE '84), 1984, pp. 362-364.

[5] G. Lawrence, *People Types and Tiger Stripes*, 3rd ed, Center for Applications of Psychological Types, Gainesville, FL, 1994.

[6] M. H. McCaulley, "The MBTI and Individual Pathways in Engineering Design," *Journal of Engineering Education*, vol. 80, 1990, pp. 537-542.

[7] 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*, vol. 55, 2000, pp. 35-42.

[8] L. Thomas, M. Ratcliffe, J. Woodbury, and E. Jarman, "Learning Styles and Performance in the Introductory Programming Sequence," proceedings of SIGCSE '02, Covington, KY, 2002, pp. 33-37.