Educational Methods for Inverted-lecture Computer Science and Engineering Classrooms to Overcome Common Barriers to STEM Student Success

A dissertation submitted in partial fulfillment of the requirements of the degree of
Doctor of Philosophy

By

KATHLEEN TIMMERMAN
M.S., Wright State University, 2012
B.S., Wright State University, 2010

2016
Wright State University
I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Kathleen Timmerman ENTITLED Educational Methods for Inverted-lecture Computer Science and Engineering Classrooms to Overcome Common Barriers to STEM Student Success BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE Doctor of Philosophy.

Travis Doom, Ph.D.
Dissertation Director

Michael Raymer, Ph.D.
Director, Computer Science and Engineering Ph.D. Program

Robert E.W. Fyffe, Ph.D.
Vice President of Research and Dean of the Graduate School

Committee on Final Examination

Travis Doom, Ph.D.

Michael Raymer, Ph.D.

John Gallagher, Ph.D.

Roger Carlsen, Ed.D.
ABSTRACT


New educational pedagogies are emerging in an effort to increase the number of new engineers available to enter the workforce in the coming years. One of the re-occurring themes in these pedagogies is variations of the flipped classroom. Often the additional classroom time gained from flipping is used to reinforce learning objectives. It is hypothesized that it might be more beneficial to students if a portion of that time is used to address common non-cognitive barriers that prevent students from succeeding in the major. In a freshman Introductory Computer Science course, three different pedagogies are compared: a hybrid lecture-active learning pedagogy, a fully flipped classroom pedagogy, and a fully flipped classroom with added barrier interventions pedagogy. All three groups are taught in SCALE-UP classrooms. While fully flipping the classroom shows a slight increase to student progression over the hybrid classroom, it is not significant. When barrier interventions are added to address motivation and interest, opportunity, psychosocial skills, cognitive skills, and academic preparedness a significant increase in student progression
occurs. Students with a low level of academic preparation are most impacted by the change. Fully-flipped classrooms with barrier interventions are implemented over the two-year core sequence for Computer Science and Engineering majors. The result shows no statistically observable change in progression rates. This provides hope that students are not just persisting through Computer Science I to fail in later courses.

The impact of the new pedagogy on under-represented female students also shows a benefit to students with a low level of academic preparation. Students from under-represented ethnic students saw the most benefit from the barrier interventions if they had a high level of academic preparedness. This suggests that these students may have been failing to progress at disproportionate rates for non-cognitive reasons, giving credence to the concept of using classroom time to address non-technical skills.
Table of Contents

1 Introduction ............................................................................................................ 1

1.1 The Need for Stem Majors ................................................................................ 1

1.2 Efforts to Increase Number of Graduates in Stem Majors ............................... 2

1.3 Dissertation outline .......................................................................................... 4

2 Background .......................................................................................................... 7

2.1 Pedagogies ....................................................................................................... 7

2.1.1 Traditional Lecture ...................................................................................... 7

2.1.2 Class Polling ................................................................................................ 8

2.1.3 Project Based Learning ............................................................................... 10

2.1.4 Activity Based Learning ............................................................................. 11

2.1.5 Flipped (or Inverted) Classrooms ............................................................... 12

2.2 Barriers to Student Success ............................................................................ 14

2.2.1 Selection of Barriers ................................................................................. 14

2.2.2 Motivation and Interest .............................................................................. 14

2.2.3 Opportunity ................................................................................................ 15

2.2.4 Psychosocial Skills ..................................................................................... 15
2.2.5 Cognitive Skill .................................................................16
2.2.6 Academic Preparation ....................................................16

2.3 Statistical Methods and Tools ........................................... 17
  2.3.1 Retained Relevant Knowledge Assessment System .......... 17
  2.3.2 Common Basic Statistical Methods ............................... 18
  2.3.3 Hierarchical Linear Modeling ....................................... 19

3 Impact of Barrier Interventions on Underprepared Computer
Science I Students ...........................................................25

3.1 Methodology .................................................................. 25
  3.1.1 Course Structure ......................................................... 26
  3.1.2 Classroom Environment ............................................. 27
  3.1.3 Control Group Structure .............................................. 27
  3.1.4 Experimental Group A ............................................... 28
  3.1.5 Experimental Group B ............................................... 28

3.2 Results ........................................................................... 31
  3.2.1 Experimental Group A ............................................... 31
  3.2.2 Experimental Group B ............................................... 32

3.3 Discussion ..................................................................... 34
  3.3.1 Term Differences in Computer Science I .................... 35
  3.3.2 Additional Support of Impact ..................................... 40
4.5.3 Retained Relevant Knowledge .................................................................................. 75

4.6 Possible sources for error .......................................................................................... 76

4.7 Student Survey on the New Pedagogy ...................................................................... 77

4.8 Conclusion .................................................................................................................. 78

5 Impact of Barrier Interventions on Underrepresented Groups of Female Students and Underrepresented Ethnic Students........80

5.1 Underrepresented Group: Female students ............................................................... 81

5.1.1 Computer Science I ............................................................................................... 81

5.1.2 Computer Science II ............................................................................................. 82

5.1.3 Computer Organization ......................................................................................... 84

5.1.4 Data Structures and Algorithms ............................................................................. 85

5.1.5 Hierarchical Linear Modeling for Female Students .................................................. 86

5.1.6 Female Progression by Academic Preparedness ....................................................... 89

5.2 Under-represented Ethnic Students ................................................................. 92

5.2.1 Computer Science I ............................................................................................... 92

5.2.2 Computer Science II ............................................................................................. 93

5.2.3 Computer Organization ......................................................................................... 94

5.2.4 Data Structures and Algorithms ............................................................................. 95

5.2.5 Hierarchical Linear Modeling ............................................................................... 96

5.2.6 Academic Preparedness Levels ............................................................................. 98
5.3 Overall Results of the Experiment ................................................................. 100

5.4 Discussion ........................................................................................................ 101

5.4.1 Gender Discussion ..................................................................................... 101

5.4.2 Underrepresented Ethnicity ....................................................................... 103

5.5 Conclusion ....................................................................................................... 104

6 Retained Relevant Knowledge Assessment System ............ 105

6.1 Motivation ......................................................................................................... 106

6.1.1 Need for Continuous Assessment .............................................................. 106

6.1.2 Goals of this Assessment System ............................................................... 106

6.2 Assessment Knowledge Topics ................................................................. 107

6.3 METHODOLOGY ............................................................................................ 110

6.3.1 Mapping coverage ..................................................................................... 110

6.3.2 Creating Direct Assessment Instruments .................................................. 112

6.3.3 Collecting the Assessment Results ............................................................. 114

6.3.4 Assessment Reports ................................................................................... 115

6.4 Database .......................................................................................................... 115

6.4.1 Database Tables .......................................................................................... 118

6.4.2 Uploading Results ....................................................................................... 122

6.4.3 Viewing Database Entries ......................................................................... 134
6.4.4 Generating Reports .................................................................................................. 135

6.5 Results & Discussion ................................................................................................. 143

6.5.1 Results and Discussion of Continuous Direct Measurements of Retained Relevant Knowledge ........................................................................................................ 143

6.5.2 Results and Discussion of Immediate Value of Assessment to Participation .......... 145

6.5.3 Results and Discussion of Overall Program Assessment ........................................... 148

6.5.4 Results and Discussion of Assessment Burden ......................................................... 152

6.6 Conclusion .................................................................................................................. 153

7 Conclusion and Future Work ....................................................................................... 155

7.1 Implements Non-Cognitive Barrier Interventions within the Classroom .............. 155

7.1.1 Future Work ............................................................................................................ 155

7.2 Increases in Progression Rates for Computer Science I Course ......................... 156

7.2.1 Future Work ............................................................................................................ 157

7.3 Exploration of Pedagogical Impact over Sequential Courses ............................... 157

7.3.1 Future Work ............................................................................................................ 157

7.4 Possible Increases in Student Progression Rate through Computer Science Core Curriculum ................................................................................................................. 157

7.4.1 Future Work ............................................................................................................ 158

7.5 Increases in Progression Rate for Under-Represented Groups ............................ 158

7.5.1 Increases in Progression Rate for Female Students with a Low Level of Academic
List of Figures

Figure 1 Ignore Classroom ............................................................................................................ 21
Figure 2 Classroom Averages ....................................................................................................... 22
Figure 3HML: Keeping classroom data without averaging ......................................................... 23
Figure 4 CS I experiment results for Spring semester................................................................. 32
Figure 5 CS I Experiment Results Summer................................................................................ 33
Figure 6 CS II Experiment Results Summer Term ................................................................. 34
Figure 7 CS I grades by term ..................................................................................................... 37
Figure 8 Academic preparedness by term for CS I ................................................................. 39
  Figure 9 Impact of flipped with barrier interventions pedagogy on different levels of academic preparedness.................................................................................................................. 40
Figure 10 Retained relevant knowledge assessment results for former Computer Science I students starting Computer Science II ............................................................ 41
Figure 11 Computer Science I Fall experiment results ............................................................ 43
  Figure 12 Progression of Computer Science I Fall students by academic preparedness level ........................................................................................................................................ 44
  Figure 13 Progression of Fall Computer Science I students with experienced instructor ................................................................................................................................. 45
Figure 14 Computer Science I Fall progression by academic preparedness with experienced instructor ................................................................. 46

Figure 15 Core Course Sequence ................................................................................................................................. 50

Figure 16 Computer Science II Results Summer Term ................................................................................................. 52

Figure 17 Computer Science II Fall Results .................................................................................................................. 54

Figure 18 Computer Science II Spring Term Results .................................................................................................... 55

Figure 19 Computer Science II Fall Term Results by Academic Preparedness ......... 56

Figure 20 Computer Science II Spring Term Results by Academic Preparedness ...... 57

Figure 21 Computer Organization Different by Semester ................................................................. 59

Figure 22 Computer Organization Results in Fall Term ................................................................................................ 60

Figure 23 Computer Organization Progression by Academic Preparedness .......... 61

Figure 24 Data Structures & Algorithms Term Progression Differences ......................... 64

Figure 25 Data Structures & Algorithms Spring Results ........................................................................................... 65

Figure 26 Data Structure & Algorithms by Academic Preparedness Level .................. 67

Figure 27 Percentage of students who enrolled in Computer Science II (1181) or both Computer Science II and Computer Organization (3310) in Sequential academic-year terms after completing Computer Science I in Spring 2015 .................................................. 71

Figure 28 Percentage of students who enrolled in Computer Science II (1181) or both Computer Science II and Computer Organization (3310) in Sequential academic-year terms after completing Computer Science I in Summer 2015 .................................................. 72

Figure 29 Percentage of students who enrolled in Computer Science II (1181) or both Computer Science II and Computer Organization (3310) in Sequential academic-year
terms after completing Computer Science I in Fall 2015 ........................................... 74

Figure 30 Retained relevant knowledge assessment for Computer Organization Spring Term................................................................................................................................. 76

Figure 31 Computer Science I Progression Rates in Fall for Female students .......... 82
Figure 32 Computer Science II Progression Rates in Fall for Female students ........ 83
Figure 33 Computer Science II Progression Rates in Spring for Female students....... 84
Figure 34 Computer Organization Progression Rates in Fall for Female students ...... 85
Figure 35 Data Structures and Algorithms Progression Rate in Spring for Female students ............................................................................................................................................... 86

Figure 36 Female Progression Rates by Academic Preparedness Levels ............... 91
Figure 37 Computer Science I Progression Rates for Under-represented ethnic students ................................................................................................................................................ 93

Figure 38 Computer Science II Progression Rates for Under-represented ethnic students ................................................................................................................................................ 94

Figure 39 Computer Organization Progression Rates for Under-represented ethnic students ................................................................................................................................................. 95

Figure 40 Data Structures and Algorithms Progression Rates for Under-represented ethnic students........................................................................................................................................... 96

Figure 41 Progression Rates by Academic Preparedness for Under Represented Ethnicities .................................................................................................................................................. 100

Figure 42 Course Progression Rates............................................................................... 101

Figure 43 Distribution of Female Students with a Low Academic Preparedness Level by
Course ........................................................................................................................................ 103

Figure 44 Mapping of Computer Science Core Courses to Knowledge Topics ........ 111

Figure 45 Mapping of Computer Science Knowledge Topics to ABET Student Learning Outcomes ........................................................................................................................................ 112

Figure 46 Sample Retained Relevant Knowledge Question ........................................ 113

Figure 47 The Database's Web Interface Homepage ...................................................... 117

Figure 48 UML of Student Focused Tables ................................................................. 119

Figure 49 UML of Meta Evaluation Tables ..................................................................... 120

Figure 50 UML of Summary Tables ............................................................................... 122

Figure 51 A portion of a retained relevant knowledge quiz results file .................... 125

Figure 52 The Database's Upload Quiz Results Webpage ....................................... 127

Figure 53 The Database Upload Internal Institute Report Webpage ....................... 132

Figure 54 An Example of a Student's View of the Retained Relevant Knowledge Quiz Result ........................................................................................................................................ 146

Figure 55 An Example Instructor View of the Results from the Retained Relevant Knowledge Assessment ........................................................................................................... 147

Figure 56 An automated Every Course Report ............................................................ 149

Figure 57 An automated Specific Course Report ......................................................... 150

Figure 58 An automated Across Program Report ....................................................... 152
List of Tables

Table 1 HLM Example Data ................................................................. 20
Table 2 ACT Math scores of CS 1180 Students ..................................... 36
Table 3 Academic Preparedness Categories ......................................... 38
Table 4 Student Survey Results: Percentage of Students Who Reported Revised Course Elements “Increased” a Success Factor ................................................................. 77
Table 5 HLM Data Summary for female students .............................. 87
Table 6 HLM Summary Data for Under-represented ethnic students ...... 97
Table 7 CS2013 Knowledge Areas ....................................................... 108
Table 8 Sample Knowledge Units in the Algorithms and Complexity Knowledge Area .......................................................................................... 109
Table 9 Sample Knowledge Topics in the Algorithms and Complexity Knowledge Area: Fundamental Data Structures and Algorithms Knowledge Unit ........................................ 109
Table 10 High Level Algorithm ......................................................... 128
Table 11 Load Quiz Technical Questions Algorithm ............................ 129
Table 12 Uploading Internal Report Algorithm .................................... 133
Table 13 Summary Report Algorithm ................................................ 136
Table 14 Get Current Course Averages Algorithm ............................... 137
Table 15 Calculate Course Graphical Information Algorithm ............. 137
Table 16 Create Course Summary Table Algorithm ............................. 139
Table 17 Get Student Learning Outcome Averages Algorithm ............ 140
Table 18 Calculate Student Learning Outcome Graphical Information Algorithm .... 141
Table 19 Create Student Learning Outcome Summary Table Algorithm ................... 142
Table 20 Breakdown Significantly Observable Differences for Course Algorithm ... 142
Table 21 Breakdown Significantly Observable Differences for SLOs Algorithm...... 143
Table 22 Courses with Deployed Retained Relevant Knowledge Assessments ........ 144
List of Equations

Equation 1 First Level Linear Regression ................................................................. 23
Equation 2 Second Level Regression Equations ......................................................... 24
1 Introduction

1.1 The Need for Stem Majors

This country may be heading towards an engineering crisis. After World War II, science, technology, and higher education helped drive the U.S. Economy by creating good jobs and successful new technology industries (Holdren & Lander, 2012). However, over the past decade the proportion of the Science, Technology, Engineering, and Mathematics (STEM) graduates has been decreasing (Holdren & Lander, 2012). This could be due to the fact that STEM majors have historically been White and Asian males (Siebens & Ryan, 2012). Now other demographics are becoming a larger portion of the overall college student body, reducing the percentage of typical STEM demographics (Siebens & Ryan, 2012). In order to maintain the current economic benefits, it is estimated that the number of STEM graduates needs to increase by approximately one million additional students over the next decade (Holdren & Lander, 2012; Lacey & Wright, 2010; Langdon, Beede, & Doms, 2011).
1.2 Efforts to Increase Number of Graduates in Stem Majors

Efforts to meet this increased demand traditionally fall into one of two categories: recruitment efforts or progression (or retention) efforts. Recruitment efforts attempt to increase the number of students that enter into STEM fields. These efforts often target underrepresented groups within the majors, especially female students, students of color, and first generation students. These student demographics typically have a higher dropout rate when compared to students who fit the traditional STEM demographics ("STEM Attrition: College Students’ Paths Into and Out of STEM Fields," 2014). Since the new students may have different academic and cultural backgrounds, they may not be as well served by pedagogies designed for the current ‘typical’ STEM student. This may result in a possible increase in dropout rates. Therefore, an increase in student recruitment does not necessarily result in a proportional increase in the number of STEM graduates. Recruiting these populations may, in fact, be a disservice to the students if they do not complete the degree.

Retention efforts attempt to increase the number of STEM majors by increasing the percentage of students who graduate with the degree. STEM fields traditionally have large dropout rates ("STEM Attrition: College Students’ Paths Into and Out of STEM Fields," 2014). These methods try to address various reasons why people withdraw from the major. Common retention efforts include co/extra-curricular activities such as developing study groups, mentorship programs, and homework help rooms. These efforts might have a greater benefit to the demographics that typically dropout at higher rates. This is the same
demographic that is commonly targeted by recruitment efforts.

There are different costs to each of these methods. Recruitment efforts generally involve marketing, awareness programs, scholarship programs, and other efforts that can be tracked as budgetary expenses. One benefit of these programs is that there is no cap on the number of additional engineers they could potentially enlist. Over time, the marginal recruitment cost per student can be tracked and optimized for a target enrollment. Generally, there is no additional commitment by or to faculty not explicitly involved in the recruitment efforts.

Similarly, the cost of retention-based efforts such as increasing academic, financial, and other support for existing students can be tracked as a budgetary expense, and, over time, the marginal retention cost per student can be tracked and optimized for a target enrollment. In these efforts, the focus is shifted from recruiting potential students who are not in the program to retaining existing students to which the program has already made a commitment. Such attempts to increase the graduation rate need not be limited to extra-curricular programs. Indeed, increases to the rate of progression of existing students may be most effectively achieved by interventions which take place within the context of the classroom. The cost of these efforts are more difficult to track, as in-classroom efforts require faculty commitment, possible faculty development, and training to help faculty adopt identified best practices. While the exact cost of implementing such efforts might be harder to define, the benefits have the opportunity to impact every student in the program without necessarily adding significant new student time burdens delivered through extra-curricular interventions. As recruiting efforts broaden the diversity of the pool of potential future STEM majors, the importance of in-classroom interventions to maintain and
increase progression rates becomes even more critical. If the pedagogy continues to primarily cater to the learning needs of the current student demographic, it should not be surprising that other groups will fail to progress at a disproportionately high rate.

The research presented in this paper focuses on retention efforts. It will specifically focus on the impact of specific, non-traditional pedagogies on student retention and progression for Computer Science and Computer Engineering majors. For decades, there has been movement towards transforming classrooms from traditional lecture to more student-centered active learning (Freeman et al., 2014). Active learning is a broad term that includes nearly any educational framework that engages the students. In some instances, this might be done by having a large project throughout the semester that the students work to complete during classroom time. It might include brief, in-class activities that test the student’s ability to apply the knowledge interspersed with the lecture. In STEM classes, it is common to see these activities and projects focus on content mastery. However, sometimes students leave the major for reasons other than cognitive ones.

Some students might struggle with progression in a major due to transitioning to the college environment. Student performance may also be impacted due to a lack of motivation and understanding of the impact the course could have on society. Some schools have started addressing some of these issues with programs such as a first year seminar course (Koch, 2000; Padgett, Keup, & Pascarella, 2013; Tobolowsky, Cox, & Wagner, 2005). However, in already tight schedules, some students opt not to take these courses.

1.3 Dissertation outline
The research in the paper examines retention efforts; specifically, active learning pedagogies specifically aimed to improve student progression. Traditionally, in engineering active learning classrooms, the activities focus on content mastery. This research hypothesizes that some classroom time may be better spent addressing non-technical issues that become barriers to students earning a degree across a two-year core sequence of Computer Science and Computer Engineering courses; I hypothesize that by including barrier interventions in the classroom, the number of students who progress through the Computer Science and Computer Engineering core sequence will increase.

Chapter 2 presents background information that helps provide context for this research. It defines vocabulary that will be used throughout the document, and it gives some information on assessment tools and statistics that will be used in later chapters.

Chapter 3 presents an introductory computer science course’s utilization of a flipped classroom pedagogy. By moving the majority of lectures to pre-recorded videos and reading assignments that could be watched by the students prior to class, additional classroom time became available for other activities. This chapter examines experimental data to gain insight on the impact of using this class time to address some non-technical barriers: motivation and interest, opportunity, psychosocial skills, cognitive skills, and academic preparation.

Chapter 4 explores the impact of the flipped pedagogy with barrier interventions that is introduced in Chapter 3 on subsequent classes in the Computer Science and Computer Engineering core sequence: Computer Science II, Computer Organization, and Data Structures and Algorithms. It discusses whether the interventions that aided in progression
through the first course are allowing students to successful progress through the program or if failure is just being postponed.

Chapter 5 explores the impact of the new pedagogy on underrepresented groups in Computer Science and Computer Engineering. The focus is on gender minorities and ethnicity minorities. It discusses if these subgroups within Computer Science are impacted differently by the flipped pedagogy with barrier interventions than their more traditional counterparts.

Chapter 6 will give more details about the Retained Relevant Knowledge Assessment System that will be summarized in chapter 2. This tool is designed to tests students retained relevant knowledge as they progress through the Computer Science and Computer Engineering curriculum. This quantitative assessment of students becomes a way of assessing student progress without fear of instructor bias or grade inflation.

Chapter 7 presents final remarks about this work of research. It also discusses areas of future work that could build upon the conclusions and discussions found throughout this dissertation.
2 Background

2.1 Pedagogies

Current efforts to increase the number of engineering majors includes recruitment efforts to attract more students and a more diverse set of students to engineering. However, with a new, more diverse population of students, it is important to re-examine traditional pedagogies to see if a different pedagogy might better server the new demographic of students.

There are an infinite number of ways to present technical content to students. Which analogies should be used? How many homework assignments should be given? What order should the material be presented? The choices the professor makes determines that professor’s pedagogy (teaching methods and practices). What follows is a brief description of several common pedagogies. These pedagogies are described as they might generally be implemented. However, every instructor will add his or her own spin to how exactly the pedagogy is implemented. Therefore, the advantages and disadvantages of each pedagogy may not hold true for every implementation of it.

2.1.1 Traditional Lecture
Historically, the traditional pedagogy is to use classroom time for a content expert to lecture to the students. The theory behind this is that an expert in the field is the best person to pass along the content to the next generation. Because of the historical precedence of this pedagogy, it tends to be the default pedagogy that instructors use. As the default pedagogy, it is also the pedagogy that most other pedagogies are compared to. However, in a Meta study performed by Freeman it is shown that almost all active learning environments perform better than the traditional lecture (Freeman et al., 2014).

Advantages:

- Most instructors and students are familiar and comfortable with this pedagogy.
- Instructor cost for classroom preparation tends to be less when compared to other common pedagogies.
- No dependence upon student preparation for class.
- No dependence upon student participation in class.

Disadvantages:

- Students appear to learn less in traditional lecture pedagogies when compared to other common pedagogies.

2.1.2 Class Polling

Class polling allows the instructor to pose a multiple choice question to the class then each student can respond using a text message or using a device such as a clicker (all students use the same method established by the instructor). The instructor receives immediate feedback on how many students selected each answer. How classroom polling
is used can vary greatly in the classroom. Some instructors just use it to gage student understanding. Other instructors grade the questions as homework to encourage student attendance and participation in the classroom. Other instructors use it for giving quizzes. Sometimes the students work in groups to answer the questions. Other times they might be responsible for coming to an answer on their own. They might be used within the traditional lecture or they might be used within the context of some other pedagogy.

Clickers are small devices that allow the students to respond to multiple choice questions within the classroom (Daniel & Tivener, 2016; Duncan, 2005; Lasry, 2008; Martyn, 2007; Mayer et al., 2009). Each student has a clicker that is registered to him or her, providing linking students to the responses. In more recent years, a shift has begun from using a clicker device to using a polling website that the students send a text message via their cell phones in order to answer the questions (Stowell, 2015). Cell phone polling option is used in the same ways as clickers, except that most have anonymity of who responded and how that person responded.

Advantages:

- Easy to use even with large class sizes.
- Integrates easily with the traditional lecture.
- Immediate feedback on student performance.
- Can be used to encourage attendance.
- Can be used to grade students

Disadvantages:

- Equipment cost and maintenance (may be minimized with cell phone option).
• Equipment malfunctions in class can be disruptive and distressing to students (especially if responses are used for grades).

• Beginning of term time cost to link students’ individual clickers to each student.

• Hard to prevent students from using a fellow classmates’ clicker when they are not present.

2.1.3 Project Based Learning

Project based learning is a pedagogy in which the students focus on one large project all term long (Becker, Member, Plumb, & Revia, 2014; Cappelleri & Vitoroulis, 2013; Correll, Wing, & Coleman, 2012). Throughout the term they are constantly adding to and modifying a solution to solve a larger problem. This pedagogy helps give a natural example of application and use, since everything is immediately applied to the Project. Project based learning can be done in two different directions: compiling or breaking apart. In a compilation style the students are typically presented with the large problem and then build the solution piece by piece. They then compile the pieces together to solve the larger problem. In a breaking apart style, the students are typically presented with a large solution that they do not understand. Then, as the term progresses, they decompose the solution into parts. As they figure out the individual parts they have already seen how it affects the larger solution.

Advantages:

• Improves students understanding when compared to traditional lecture

• Directly connects content to real world application
• Shows how implementations integrate into a larger solution

Disadvantages:

• High initial cost of designing the project.

• Complexity of the large project may be overwhelming to students

• Failure to solve/understand previous piece of the project could affect student ability to solve/understand the next piece.

2.1.4 Activity Based Learning

Activity based learning is a pedagogy that is very similar to project based learning. The difference is rather than focusing on one large project, the students solve many smaller problems (Beicher & Saul, 2003; Beichner, Saul, Allain, Deardorff, & Abbott, n.d.; Gannod, Burge, & Helmick, 2008; Reddy, Mishra, Ramakrishnan, & Murthy, 2015). By having many problems, the students get to see how the classroom content applies to many different areas. There is also less dependence on previous understanding and solutions for solving the current problem. The disadvantage of activity based learning when compared to project based learning is that the students typically do not get as good of an understanding on how smaller solutions compile into larger solutions. Also, if the activities are not presented in the context of a larger problem, the solutions might appear trivial and non-relevant to real world problems.

Advantages:

• Improves students understanding when compared to traditional lecture

• Can connect content to real world problems
• Less dependence on previous solution and understanding for solving current problem when compared to project based learning

• Easy to include a wide variety of applications for content

Disadvantages:

• High initial cost of designing the activities.

• Students will generally not obtain as good of an understanding of how all the components combine as when compared to a project based learning pedagogy.

• If problems not placed in context, problems might appear trivial or non-relevant

2.1.5 Flipped (or Inverted) Classrooms

The flipped (or inverted) pedagogy takes the traditional lecture homework style and flips it over. The students are given the lecture to do on their own and do the “homework” in the classroom. The lecture traditionally is given in the form of reading material (textbooks, articles, etc.) and/or videos that the students are required to watch before entering the classroom (Azemi, 2013; Beicher & Saul, 2003; Beichner, Saul, Abbott, et al., n.d.; Bergmann & Sams, 2012; Gannod, 2007; Lage, Platt, & Treglia, 2000; Mason, Shuman, & Cook, 2013; Rhodes, 2016; Strauss, 2012; Toto & Nguyen, 2009; Veen & Karls, 2013). Then the students come to class, and the instructor uses some of the time previously dedicated to one-to-many broadcast lecture to more actively involve students in the application and synthesis of that knowledge. It is common to make these activities that the students complete in small groups. Then as students come across issues while completing the activities, they have the instructor immediately available to clarify any misconceptions.
Advantages:

- Improves students understanding when compared to traditional lecture
- Integrates well with either project based learning or activity based learning
- Helps prevent students from building bad habits since the activities are completed in the instructors presence

Disadvantages:

- High initial cost of finding quality lecture material
- High initial cost of developing in class activities
- Instructors might unintentionally add extra work for students by not reducing or removing homework to compensate for the time needed to review lecture material

All of these different pedagogies have both advantages and disadvantages. However, Freeman’s research shows that almost all styles of active learning outperform the traditional lecture (Freeman et al., 2014). There is currently a push to move more classrooms to an active learning pedagogy (Holdren & Lander, 2012). However, there is currently limited data on how active learning pedagogies compare to each other. This could be due to the wide variety of styles within even a single category of pedagogy. Most of the pedagogies in STEM classrooms use the active learning time with working problem. This seems straight forward as application is often an important objective. However, sometimes the obstacles that prevent students from progressing through a major are not cognitive barriers. This research proposes that a portion of this active learning classroom time could
be used to specifically address common barriers to student success.

2.2 Barriers to Student Success

2.2.1 Selection of Barriers

Every group of students struggles against different sets of barriers to progression in their chosen program of study (Duckworth & Yeager, 2015). For this study, we consider the general applicability of barriers initially identified as barriers to success for Students with Disabilities. The Ohio’s STEM Ability Alliance program has demonstrated a 90% progression rate for students with disabilities participating in a set of extra-curricular interventions to overcome the set of identified barriers (Shingledecker, Auld, Todd, & Weibl, 2014). We hypothesize that many of the same barriers may impact other at-risk groups in the Computer Science and Engineering program. The five barriers addressed in our research methods are motivation and interest, opportunity, psychosocial skills, cognitive skills, and academic preparedness.

2.2.2 Motivation and Interest

Excluding traditional academic achievement scores The College Board found that the major non-cognitive factor that predicts college success is motivation and interest in the topic (The College Board, 1997). Motivation and interest is measured by persistent effort in school and out-of-class intellectual activities. Early motivational career awareness interventions positively affect relevant student behaviors including academic planning, improved math and science grades, and advanced course enrollment (Fouad, 1995). High school seniors more motivated about their vocational goal, are more focused, ready, and
likely to complete postsecondary training than those unmotivated about their career goals (Schroedel, 1991). Additionally an increase in relationships between a student and his or her peers as well as the instructor can lead to the student valuing the degree more (Iii, Williams, & Strayhorn, 2013). Students who do not identify with the major are less likely to pursue it (Lord, Member, Layton, & Ohland, 2015), and students who do pursue it might be motivated in different ways depending on gender (Jaynes & Cummings, 1963).

2.2.3 Opportunity

Inadequate educational and employment opportunities continue to be major barriers for disabled persons, as well as members of other minority and disadvantaged groups. While they have been reduced by education and policy actions, barriers created by negative employer attitudes and inaccurate knowledge still limit job opportunities (Dixon, Kruse, & VanHorn, 2003; National Organization on Disability, 2004). More subtle barriers to opportunity commonly shared by disabled and other students are created by teachers who harbor doubts about their academic potential (Eddy, Brownell, Thummaphan, Lan, & Wenderoth, 2015; Eddy, Brownell, & Wenderoth, 2014; Grunspan et al., 2016; Wagner, Kutash, Duchnowski, & Epstein, 2005), and by low parental expectations for success in postsecondary education (Wagner et al., 2005). Both factors are reflected in behaviors that can subtly guide students toward other pursuits and discourage growing talents.

2.2.4 Psychosocial Skills

Acquiring self-determination and self-advocacy skills is essential for all students, especially those with life experiences that can affect self-image and self-efficacy. Good
decision making for these students involves awareness, and proper exertion of their rights and responsibilities. The student’s ability to determine and explain what their needs are in regards to their education is of the utmost importance (Eckes & Ochoa, 2005; Schutz, 2002). Despite their critical role for at risk students, teachers indicate a need for better training and materials to promote the development of effective interpersonal interaction skills and attitudes that result in perseverance in the face of obstacles and challenges (Schutz, 2002).

2.2.5 Cognitive Skill

Academic performance in post-secondary STEM education is fundamentally shaped by cognitive skills (Ruban & McCoach, 2005). Essential cognitive skills including communication, problem solving, reasoning, decision making and critical thinking are believed to underlie nearly all academic achievement, especially in areas associated with the scientific process and mathematics. The full development of these skills is limited for economically disadvantaged students that may have experienced mediocre primary and secondary educational experiences, especially in the areas of scientific inquiry and mathematics. The lack of opportunities to develop strong thinking skills are common in students who often get steered away from cognitively challenging course work by circumstance and subtle cultural influences.

2.2.6 Academic Preparation

Evidence supporting the belief that student postsecondary and career success is rooted in rigorous K-12 academic preparation and the early influences of families, teachers, and
cultural factors in the student’s life is incontrovertible (Buckley, Bridges, & Hayek, 2006; Perna & Thomas, 2006; Tierney, Corwin, & Colyar, 2005). Sociological impediments to access and learning that are associated with underrepresented groups create serious barriers to crucial academic preparation in the demanding areas of science and mathematics as well as reading and writing.

2.3 Statistical Methods and Tools

In STEM education research, it is common to categorize by letter grades. Letter grades of A, B, and C are considered passing. Letter grades of D, F, and W are considered failing grades as often these grades do not allow students to progress to sequential classes. These letter grades may come from any relevant graded source (quiz, exam, homework assignments, in class activities, labs, etc.). For the context of this paper, unless otherwise stated, a grade represents the final course grade.

2.3.1 Retained Relevant Knowledge Assessment System

The Retained Relevant Knowledge Assessment System is tool developed to provide a direct measurement of student performance throughout the program. The exact details of this assessment tool can be found in Chapter 6. A limited summary is included here.

This assessment system maps core curriculum courses learning objectives to the Association for Computing Machinery and IEEE-Computer Society joint tasks forces knowledge topics recommended as program standards for Computer Science and Computer Engineering programs (The Joint Task Force on Computing Curricula - ACM/IEEE-Computer Society, 2013, 2016). Each course mapped knowledge topics that
the course requires prior to beginning the class (prerequisite knowledge) and to the knowledge topics that would then be developed within that course.

Using the mapping of knowledge topics to courses, a prerequisite quiz is developed to test incoming students on the knowledge topics that instructors assumed the students have learned in prior courses. The question creation is distanced from common instructors to tie mastery to national (and not instructor by instructor) based standards. The quiz is given to all students in the course at the beginning of the course through an online assessment system (Desire2Learn, 2016).

Each term, the results from all the course quizzes are collected and stored within a database to be used to assess the overall program effectiveness. The database also collects basic demographic information on the assessed students. Additionally, at the end of the term, administrators run an internal institutional report to collect data on final grades for the students. This report is added to the database. The database can then be used to gain information on various items. This report specifically uses the database information to evaluate how students in the new pedagogy classrooms perform on these standardized questions compared to their control counterparts.

2.3.2 Common Basic Statistical Methods

In STEM education research it is common to see many of the common basic statistical methods. Difference of means is often used to compare the grades in one pedagogy to another pedagogy. These tests are done with z-tests, t-tests, or Mann-Whitney U tests. It is also common to see other standard tests such as Pearson correlations and ANCOVA tests.
2.3.3 Hierarchical Linear Modeling

Hierarchical Linear Modeling (HLM) is a regression model that maintains information regarding data combination at different levels while completing a variation of an ordinary least squares regression for testing variance (Woltman et al., 2012). Consider the following example presented in Woltman’s tutorial where breakfast consumption is being compared to a GPA score of some students (Woltman et al., 2012). Students’ classroom, GPA score, and breakfast consumption are recorded (See Table 1).
**Table 1 HLM Example Data.**

This table is a variation from Woltman [45].

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Classroom ID</th>
<th>GPA Score</th>
<th>Breakfast Consumption Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

When classroom ID is ignored and breakfast consumption is graphed against GPA score, Figure 1 is produced. This figure shows a slightly negative regression.
This figure is from Woltman (Woltman et al., 2012). The figure shows each student's breakfast consumption score versus his or her GPA score. The overall trend is a slightly negative regression.

In an attempt to keep information about the classroom, each classroom could be averaged and then the averages could be plotted (Figure 2). This results in another negative regression.
Figure 2 Classroom Averages

This figure is from Woltman (Woltman et al., 2012). In this case, in an attempt to keep information about classrooms, the classroom GPA scores and breakfast consumption scores have been averaged. The result is a more pronounced negative regression.

HML considers both the regression between groups and within groups resulting in a positive regression (See Figure 3) (Woltman et al., 2012).
**Figure 3** HML: Keeping classroom data without averaging

This figure is from Woltman (Woltman et al., 2012). HML attempts to keep additional information by using a variation of a least square means regression.

HLM is solved through a series of equations. First, a linear regression is performed on each sub group at the lower level. (In the provided example, this step would be running a regression on each classroom). Equation 1 is used for this.

*Equation 1 First Level Linear Regression*

\[ Y_{ij} = \beta_{0i} + \beta_{1i}X_{ij} + r_{ij} \]

\( Y_{ij} \) is the dependent variable. \( X_{ij} \) is the level-1 predictor. \( \beta_{0i} \) is the intercept for the \( j \)th level-2 predictor. \( \beta_{1i} \) is the regression coefficient associated with the \( X_{ij} \) for the \( j \)th level-2 unit. The random error associated with the \( i \)th level-1 unit nested within the \( j \)th level-2 unit is \( r_{ij} \).

After this regression is performed, another regression is performed using an offset for each level two category.
Equation 2 Second Level Regression Equations

\[
\beta_{0i} = \gamma_{00} + \gamma_{01} G_i + U_{0i}
\]

\[
\beta_{1i} = \gamma_{10} + \gamma_{11} G_i + U_{1i}
\]

\(\beta_{0i}\) is the intercept for the \(j\)th level-2 predictor. \(\beta_{1i}\) is the regression coefficient associated with the \(j\)th level-2 unit. \(G_i\) is the offset associated with the level-2 predictor. \(\gamma_{00}\) is the overall mean intercept adjusted for \(G\). \(\gamma_{10}\) is the overall mean intercept adjusted for \(G\). \(\gamma_{01}\) is the regression coefficient associated with \(G\) relative to level-1 intercept. \(\gamma_{11}\) is the regression coefficient associated with \(G\) relative to the level-1 slope. The random effects associated with the \(j\)th level-2 unit is \(U_{0i}\) when adjusted for \(G\) on the intercept and is \(U_{1i}\) when adjusted for \(G\) on the slope.

As a linear regression model, Hierarchical Linear Modeling makes all the common assumptions for linear regression models. These assumptions are: \(E(U_{0j}) = 0; E(U_{1j}) = 0; E(\beta_{0j}) = \gamma_{00}; E(\beta_{1j}) = \gamma_{01}; \text{var}(\beta_{0j}) = \text{var}(U_{0j}) = \tau_{00}; \text{var}(\beta_{1j}) = \text{var}(U_{1j}) = \tau_{11}; \text{cov}(\beta_{0j}, \beta_{1j}) = \text{cov}(U_{0j}, U_{1j}) = \tau_{01}; \text{cov}(U_{0j}, r_{ij}) = \text{cov}(U_{1i}, r_{ij}) = 0.\)

Depending on the implementation of the algorithm, \(G\) may be assigned or calculated. In this research, the lmer library for R is used. This library uses a model-fitting algorithm that estimates the population distribution from which the \(G\) effects are drawn. The assumptions for this estimation are a normal distribution, a mean of 0, and a variance of \(\tau_{00}\).
3 Impact of Barrier Interventions on Underprepared Computer Science I Students

A flipped classroom pedagogy is used in an introductory computer science course. By moving the majority of lectures to pre-recorded videos and reading assignments that could be watched by the students prior to class, some additional classroom time is freed up for other activities. This chapter examines experimental data to gain insight on the impact of using this class time to address some non-technical barriers: motivation and interest, opportunity, psychosocial skills, cognitive skills, and academic preparation.

3.1 Methodology

This experiment will compare three different pedagogies effectiveness in a Spring Semester Computer Science I course. This course has the following objectives:

- fluency in a programming language
- competency in the object-oriented paradigm
• competency in the event-driven programming paradigm
• ability to communicate effectively in a programming language with a focus on style towards developing increasingly self-documenting high-level code

The three pedagogies all incorporate active learning to varying degrees. The control pedagogy is a hybrid lecture-active learning environment where roughly half the time is spent in lecture and roughly half the time is spent on activities. Experimental Group A is in a fully flipped environment where lectures have been moved outside the classroom and classroom time is spent largely on activities and post processing activities that directly improve understanding of course material. Experimental Group B is similar to Experimental Group A with the exception that some classroom time is intentionally spent completing activities that help overcome common cognitive and non-cognitive barriers to student success. There are two different instructors for the control group and Experimental Groups A and B have different instructors as well. While this does lead to a potential for experimental error, all the instructors have prior experience teaching this course content to this level of student.

3.1.1 Course Structure

This experiment is run across multiple sections of the course and, with a limitation on available classroom space, some of the sections are taught in a Monday, Wednesday, Friday structure while others are taught in a Tuesday, Thursday course structure. However, the contact hours each week remains the same. Additionally, all classes are taught in the SCALE-UP classroom (described in Section 3.1.2), but some are taught in a 36 person classroom and others are taught in a 54 person classroom. The larger room has a teaching
assistant to compensate for the increase in the number of students. The students did not know in which style of pedagogy they are enrolling.

In addition to classroom time, the students from all sections are required to sign up for a mixed-section common lab experience that met twice a week for one hour each time. The students have common weekly labs (twelve total) and common projects that are due every two weeks (six total). These items make up the majority of the grade, and are graded by teaching assistants who did not know which pedagogy their students have for lecture.

3.1.2 Classroom Environment

Both the control group and the experimental groups in this study are using a Student Centered Active Learning Environment for Upside-down Pedagogies (SCALE-UP) classroom environment (Beicher & Saul, 2003). SCALE-UP is one of many active learning approaches. A SCALE-UP classroom integrates the use of technology to aid in learning. Students sit at tables facing each other, rather than the instructor. Typically, students work in small groups of roughly three students. Groups may combine into larger groups of six to nine students throughout the activities. Screens and whiteboards around the classroom allow the groups to collaborate and display their work. Student exercises are designed to leverage the technology resources in the SCALE-UP room to enhance student participation, and inter-group sharing of results (Beichner, Saul, Allain, et al., n.d.; Robert J. Beichner, 2008).

3.1.3 Control Group Structure

The control group uses a hybrid traditional-active learning structure and is held in a
SCALE-UP classroom. The students are assigned to read the textbook prior to coming to class. Then when they arrived at class, they would take a two to five point quiz (340 points total for 17 percent of the final grade). After that, the instructors would spend approximately half the classroom time lecturing and half the classroom time completing classroom activities that improved technical skills. In addition to the labs and projects previously described in the course structure section, the students have two midterms and a final exam.

3.1.4 Experimental Group A

The Experimental Group A is held in a SCALE-UP classroom. The lecture is moved to textbook reading assignments, giving more time for additional activities during the classroom time. The students are required to complete a take home quiz over the reading assignment. Focused videos (used in Experimental Group B) are available but not emphasized or required.

Once the students attended class, they would begin each class period with an active-learning activity. At the end of the week they would take a short quiz over that week’s material. These pre- and post- quizzes replaced the midterms. The students still have a final exam, labs, and projects.

3.1.5 Experimental Group B

The Experimental Group B is held in a SCALE-UP classroom. To make additional time to include more activities and interventions, the lecture is moved to focus videos that the students watched before coming to class. The students are assigned to watch several short
videos (normally less than 10 minutes long each) prior to attending each class. The total time assigned for videos is generally kept under 25 minutes per class period. Then once the students attended class, they would begin class period with an active learning activity. Some of these activities would simultaneously directly or indirectly address an identified barrier. For each barrier, at least two interventions are implemented inside the classroom during the academic term. Each class session, students would be given a short quiz (approximately four multiple choice questions) testing them over the knowledge they have learned from the previous day’s activities. These quizzes replaced the midterms. The students still have a final exam, labs, and projects.

Below is examples of the types of barrier interventions that are used for Experimental Group B.

3.1.5.1 Motivation and Interest

To help keep students motivated and interested in computer science and engineering, time is spent defining computer science, what a computer scientist does, and how computer scientists are different from general programmers. Real world problems are used such as gene identification in bioinformatics, and simulating dice rolls for games. Different jobs within computer science and engineering are discussed. Additionally, the student projects are made more open ended, so the students have an opportunity to apply their skills and cultural competencies to problems that most interested them.
3.1.5.2   Opportunity

The students are made aware of job opportunities related to the course material and the required preparation for those careers. Also, while completing classroom activities, students have a chance to realistically compare themselves to their peers. The students are also informed about what would likely happen during the interview experience. They are given a chance to collaboratively develop the communication, technical, and professional skills within the classroom activities.

3.1.5.3   Psychosocial Skills

Some classroom time is spent addressing cognitive bias and cultural competency issues from (Greenwald & Nosek, 2015; Smyth, Nosek, Greenwald, & Nosek, 2015). Also, classroom activities helped develop interpersonal interaction skills. The previously mentioned interview preparations and realistic peer comparisons also could help overcome this barrier to student success.

3.1.5.4   Cognitive Skills

The classroom group activities worked as interventions to cognitive skills. The students worked in teams to improve communication. They are given problems to solve. They then have to discuss with their teammates different approaches and use reasoning to decide which method to implement. Additionally, they would get to compare and contrast their solution to other solutions, developing critical thinking skills.
3.1.5.5 Academic Preparation

The students are given the lectures as pre-recorded videos. This allowed students to pause, rewind, and re-watch the videos as needed. They are also given daily quizzes that tested the previous day’s knowledge. This gave the students immediate feedback on how well they have learned the material, and gave them incentive to not fall behind, by waiting on an exam date to cram.

3.2 Results

The student performance of computer science and engineering majors is examined for this project. Three different pedagogies are compared for an introductory course, Computer Science I (CS I). Students in this course need an A, B, or C to progress on to the next course.

3.2.1 Experimental Group A

For Experimental Group A, CS I is flipped using textbook reading assignments, but no barrier interventions are consciously included for 56 students. Figure 4 shows a slight increase in progression rate (53% to 59%). However, this increase did not test significant in a one-tail hypothesis test with a 95% confidence value. The resulting p value is .29. Nevertheless, removing the lecture from the classroom did not appear to cause issues for the freshman students.
When the classroom is fully flipped with experimental group A, a slight increase occurred. This increase did not test significant. When the classroom is fully flipped and barrier interventions are used 75% of the students are able to progress to the next course.

3.2.2 Experimental Group B

In Spring 2015, CS I is flipped using this model of barriers interventions for 28 students. CS I saw a significant increase with a 95% confidence in a one-tail hypothesis test in the number of students who passed the class at the level that they are able to progress to the next course in the sequence (ABC): 53% to 75% (Figure 4). The p value for this test is .016.

The experiment is repeated in the Summer term for 16 students with a different instructor. Summer structure is slightly different in that lab meets only once a week, but is twice as
long. Also Summer term is only 12 weeks rather than 14 weeks long. Class time is increased slightly to reach the needed number of contact hours as Fall and Spring terms. The content, lab, projects, quizzes, midterms, and final exams remain consistent. In this instance, every Computer Science or Computer Engineering major who took the course received a passing grade Figure 5. While it is not expected to always have 100% progression, this difference is substantial and significant. A one-tailed hypothesis test resulted in a $p$ value of 0.0047.

![Figure 5 CS I Experiment Results Summer](image)

When the fully flipped with barrier interventions is implemented in the Summer term, there is a substantial and significant increase in the number of students who are able to progress to the next course.

The experiment is also repeated in the Summer term for 18 students with yet another
instructor for the sequential course CS II. CS II has the same structure as CS I. Figure 6 shows an increase from 80% progression to 89% progression that did not test significant in a two-tailed hypothesis test. The $p$ value is .29.

![Pie charts showing progression rates]

**Figure 6 CS II Experiment Results Summer Term**

When the fully flipped with barrier interventions is implemented in the Summer term, there is a statistically unobservable increase in progression rate to the next course.

### 3.3 Discussion

In this experiment, three different classroom pedagogies are examined. The control group is a hybrid traditional-active learning environment in which roughly half the time is spent on lecture and half the time is spent solving problems. Experimental Group A used
a fully flipped classroom, and Experimental Group B used a fully flipped environment with some barrier intervention.

Removing the lecture did not result in a statistically observable improvement. However, with the barrier interventions there is a significant improvement. The student progression rate increased by almost 50%. This 75% progression rate has a secondary interesting fact in how it relates to the Fall Semester progression rates.

3.3.1 Term Differences in Computer Science I

The traditional progression in CS I varies depending on the semester. The Fall semester generally outperforms the Spring semester as the students most academically well-prepared when entering college are advised to take the course in the Fall semester of their freshman year. The Spring semester tends to be a combination of students who are academically not prepared for the course in the Fall (and therefore have to take a prerequisite in the Fall before being admitted to the course for the Spring), students who dropped, failed, or withdrew Fall semester (DFW), students who switched majors, and non-traditional students. The difference between Fall and Spring semesters’ progression rates is significant when a tested using a one-tailed z-test with $\alpha$ of .05. The resulting $p$ value is 0.0026. The average progression rate for Fall semester is 68% while Spring semester is only 53%. This difference in progression rate could be due to the difference in academic preparation of the students entering the course. The difference in academic preparation between Fall and Spring semester is also significantly different. A one-tailed z test with $\alpha$ of .05 resulted in a $p$ value of 0.0057 (See Table 2).
### Table 2 ACT Math scores of CS 1180 Students

<table>
<thead>
<tr>
<th></th>
<th>Fall</th>
<th>Spring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>26.00977</td>
<td>24.87963</td>
</tr>
<tr>
<td>Standard Error</td>
<td>0.180869</td>
<td>0.4084</td>
</tr>
<tr>
<td>Median</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>Mode</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>4.092605</td>
<td>4.244221</td>
</tr>
<tr>
<td>Sample Variance</td>
<td>16.74942</td>
<td>18.01341</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>0.512618</td>
<td>-0.0534</td>
</tr>
<tr>
<td>Skewness</td>
<td>-0.4538</td>
<td>-0.13685</td>
</tr>
<tr>
<td>Range</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Minimum</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Maximum</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>Sum</td>
<td>13317</td>
<td>2687</td>
</tr>
<tr>
<td>Count</td>
<td>512</td>
<td>108</td>
</tr>
</tbody>
</table>

Hypothesized Mean Difference

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>2.530197</td>
</tr>
<tr>
<td>P(Z&lt;=z) one-tail</td>
<td>0.0057</td>
</tr>
<tr>
<td>z Critical one-tail</td>
<td>1.644854</td>
</tr>
</tbody>
</table>

The progression for Experimental Group B (flipped with barrier interventions) is 75% which is more typical of a Fall Semester class even though it is completed in the Spring.
term. Figure 7 charts this difference in the student progression by term.

Historically, Spring term students progress at lower rates than their Fall term counterparts. This is not unexpected as students entering academically prepared to take the course, generally take it fall term. Students who are not academically prepared are required to take a prerequisite and then generally take the course in the Spring, along with students who failed or withdrew from the course in the Fall.

The difference of impact on different levels of academic preparedness for the control group and Experimental Group B is explored further by binning the students by academic preparedness levels. The students are given a level of academic preparedness of high, medium, or low. These levels are based on Math Placement Scores and ACT Math scores.

Figure 7 CS I grades by term

The diagram shows a comparison of CS I grades by term for different groups. The bars represent the percentage of students receiving each grade (A, B, C, D, F, W) for each term from Fall 2012 to Spring 2014. The Experimental Group B: Fully Flipped with Barrier Interventions is highlighted.
The scores are categorized as follows:

*Table 3 Academic Preparedness Categories*

<table>
<thead>
<tr>
<th></th>
<th>MPL</th>
<th>ACT-Math</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>7 plus</td>
<td>28 plus</td>
</tr>
<tr>
<td>Medium</td>
<td>5 to 6</td>
<td>25 to 27</td>
</tr>
<tr>
<td>Low</td>
<td>4 or less</td>
<td>24 or less</td>
</tr>
</tbody>
</table>

If a student fell into two different categories, they are placed in the higher category. For example, a student with a Math Placement Level of 6 and an ACT-Math score of 29 would be categorized as having a “High” level of academic preparedness. The categories are created to divide Fall semester CS 1180 students into the three categories as evenly as possible.

Figure 8 charts the difference number of students in each academic preparedness category for both terms. The Fall term has less than 30% of its students in the low category. The Spring term has roughly half of its students falling into the low level of academic preparedness.
Figure 9 breaks down the impact of Experimental Group B by the different levels of academic preparedness. The difference for high and medium levels of academic preparedness have too few students to justify commenting on them. For the low level of academic preparedness students, there is clearly a significant increase in the number of students who progressed.

Figure 8 Academic preparedness by term for CS I
This Figure bins students based on academic preparedness levels. In the Fall semester, the bins are close to balanced, with the smallest bin being the low level of academic preparedness bin. In the Spring semester, roughly half of the students are in the low level of academic preparedness bin.
3.3.2 Additional Support of Impact

When the experiment is repeated in the Summer term for Experimental Group B, with the instructor who had taught Experimental Group A, she too saw a significant increase in the number of students who progressed (See Figure 5). This lends credence that it might not just be a difference in instructor that resulted in the increased progression rate.

When the sequential course is taught the next semester using the same structure as Experimental Group B, the result is an increase from 80 percent progression to 89 percent progression that did not test significant. This is important as the class is almost entirely made up of students who have previously failed the course or students who have taken CS
I as part of an experimental group the previous Spring. Therefore, this result gives hope that the increased progression is not gained at the expense of decreasing the standards.

3.3.3 Retained Relevant Knowledge Assessment

As a way of completing continuous assessments of student retained knowledge, a quiz of relevant prior learned knowledge is given to the students in Computer Science II at the beginning of the term. Chapter 6 contains more details on the collection of this data.

![Figure 10: Retained relevant knowledge assessment results for former Computer Science I students starting Computer Science II](image)

*Figure 10* Retained relevant knowledge assessment results for former Computer Science I students starting Computer Science II

Every student who took the retained relevant knowledge assessment quiz answered 15 questions. This figure displays the percentage of questions answered correctly versus the percentage answered incorrectly for each of the three experimental groups. Standard error shown.

Figure 10 charts the overall results of the retained relevant knowledge assessment. The
chart shows the percentage of questions answered correctly for each group of students. Experimental Group B: Flipped with Barrier Interventions increased progression without significantly decreasing scores on the retained relevant knowledge assessment given at the beginning of the course. This again supports the idea that the students did not progress through Computer Science one at a higher rate by lowering the expectations of the students.

3.4 Results for Standard Track Computer Science I Students

Experiment B, the fully flipped with barrier interventions, is repeated again during the following Fall semester. As previously discussed in Section 3.3.1, there is a difference between Fall and Spring semester students. Figure 8 showed that Fall semester has a more balanced academic preparedness levels versus the unbalanced levels of Spring semester. When the experiment is completed in the Fall, it followed the same methodology as the Spring Experiment Group B (See Section 3.1). The experiment is run using three different instructors. The overall results are shown in Figure 11. There is slight decrease between the control students that are in the hybrid traditional-active learning environment and the experimental group. This decreased did not test significant in a one-tailed z test with a 5% α. The resulting p value is .372. The n values are 397 and 141 for the Control and Experimental groups respectfully.
The results are broken down by academic preparedness levels explained in Section 3.3.2. These results can be seen in Figure 12. When viewed by academic preparedness level, the new pedagogy is detrimental to student progression, especially for students with a medium level of Academic Preparedness. However, as previously stated, there are three different instructors for fall semester, and only one of them has previous experience teaching the Computer Science I course. Difference in instruction may contribute to this change in results.

**Figure 11 Computer Science I Fall experiment results**
This figure illustrates the impact of the fully flipped with barrier interventions pedagogy on the traditional track students who take the course during the Fall semester.
When the section for the single instructor who has previous experience with the course data is pulled, the results still did not test significant in a one-tailed z-test with 5% \( \alpha \) (See Figure 13). The resulting \( p \) value is .32. The \( n \) values where 397 and 81 for the control and experimental groups respectfully.

**Figure 12 Progression of Computer Science I Fall students by academic preparedness level**

This figure illustrates the percentage of students who progressed forward through Computer Science I in the Fall semester. The hybrid traditional-active learning control group (HLA) and the fully flipped with barrier interventions (FI) groups are shown. Standard error shown.
When the results for the single previous experience instructor are broken down by academic preparation level, an increase is shown for the less academically prepared (See Figure 14). This increase for the less academically prepared matches the results seen in Figure 9.

Figure 13 Progression of Fall Computer Science I students with experienced instructor

This figure illustrates the impact of the fully flipped with barrier interventions pedagogy on the traditional track students who take the course during the Fall semester, but only considers students from the experienced instructor in the Experimental group.
3.5 Possible sources for error

Any time experiments are completed with people, there is a certain amount error that will occur simple because there is no way to control for all variables surrounding a person. Things going on in people’s personal life could be reflected in their academic performance for both the instructors and the students. Some groups of students might study for more hours or more intently when they study. Some students might know better test taking strategies or more effective study skills. For this reason, there is a lot of room for
experimental error in Educational Research. While a large n value can help normalize this impact, not all the experiments presented here have a large n value, especially when the group is divided into sub groups as it is done for academic preparedness. So while results are promising, they must also be viewed with a level of caution.

In addition to the human factor element, this experiment has other factors that could have led to experimental error. This experiment is implemented over the entire Computer Science and Computer Engineering core sequence; therefore, it is impractical to have one instructor for all the sections. For Computer Science I the control is made up of two different instructors and Experimental Groups A and B have different instructors as well. While the instructors are different they all have experience with teaching this material at this level. The fact that the same instructor is used for Experimental Group A in the Spring term (in which the increase in progression did not test significant) and Experimental Group B in the Summer term (in which the increase in progression did test significant) gives hope that the resulting increase is not just instructor dependent.

This experiment is conducted on Computer Science and Computer Engineering students. The Control Groups are comprised of students who have taken the course in previous years. If a student took the course as a Computer Science or Computer Engineering major, but then switched majors before the data is collected for the control group, that data would not have been available. As students who are failing to progress through courses are at a higher risk to change majors, this could make the Control Groups numbers appear stronger than they actually are.

3.6 Conclusion
A fully flipped classroom with barrier interventions, some of which addressed non-cognitive barriers, significantly increased progression rates in a first year Computer Science course. The students with a low levels of academic preparedness have the largest increase in progression rate. This increase in progression does not appear to have occurred by lowering standards for the students. The progression rate increase for the fully flipped with barrier interventions pedagogy makes a traditionally less prepared class perform at rates more in line with their more academically prepared counterparts.

When the classroom is fully flipped but did not have the barrier interventions an insignificant increase in progression rate occurs when compared to a hybrid traditional-active learning pedagogy.
4 Barrier Interventions throughout the Computer Science and Engineering Core Sequence

The flipped pedagogy with barrier interventions that are introduced in Chapter 3 increased the progression of less academically prepared students in the first core Computer Science course. This chapter explores the same pedagogy used in the next three subsequent core courses: Computer Science II, Computer Organization, and Data Structures and Algorithms (See Figure 15). It explores whether the additional students progressed through Computer Science changes the progression rates of the subsequent core courses. The interventions that aided in progression through the first course combined with no change in subsequent course progression rates would result in more students’ successful progress through the program.
Computer Science II is a course that is traditionally taken in the second half of freshman year. It is the second course in the core sequence of Computer Science and Computer Engineering curriculum (See Figure 15). This course has the following objectives:

- competency in recursion and recursive programming
- competency in fundamental data structures and algorithms
- ability to read, reuse and extend high-level code
- competency in analyzing problem requirements and developing program design
- understanding computation cost associated with alternative designs ability to understand and apply defensive programming techniques

4.1.1 Methodology Computer Science II
The experiment for the Computer Science II course is designed the same way as the experiment for the Computer Science I course described in Section 3.1 but it only uses the flipped with interventions pedagogy. There is no inclusion of the flipped without intentional barriers in any of the remaining experiments. The control group is structured the same as the previous control group: a hybrid traditional lecture-active learning environment as described in Section 3.1.3. The control group is made of several different instructors. The experimental group is a fully flipped environment where lectures have been moved outside the classroom and classroom time is spent largely on activities and post processing activities that directly improve understanding of course material. Additionally, the experimental group spends some classroom time intentionally completing activities that help overcome common cognitive and non-cognitive barriers to STEM student success. The experimental group also has different instructors. More details on methodology can be found in Section 3.1. Details about the experience and number of different instructors is provided with the results.

4.1.2 Results for Computer Science II Experiment

The experiment is conducted over three different terms: Summer 2015, Fall 2015, and Spring 2016. The first experimental group is during the Summer 2015 term. There are 18 students taking Computer Science II this term taught by a single instructor who has not previously taught the course. This group of students is largely made up of students who have previously failed the course or students who participated in the Experimental Groups A and B from the experiment described in Chapter 3. The control group is made up of Computer Science II students from all three terms from Fall 2012 through Spring 2015.
The $n$ value is 149 for the control. The control group is made up of many instructors, most of who have prior experience teaching the course. There is a slight increase in progression (80% to 89%) but this slight increase did not test significant ($p = .29$). The shift in grades is not statistically observable either (two-tailed t-test, $p = .46$). Figure 16 illustrates the results.

![Graph showing results](image)

**Progress (ABC) = 80%**  
**N = 149**  

**Progress (ABC) = 89%**  
**N = 18**  
Not significant $p = .29$

Figure 16 Computer Science II Results Summer Term
When the fully flipped with barrier interventions is implemented in the Summer term, 89% of the students are able to progress to the next course.

This experiment is repeated in the Fall 2015 term. Fall term typically has a bi-modal distribution of students. This is due partially to the non-traditional track nature of the course when taking Computer Science II in the Fall. Students who come to campus less academically prepared are required to take a prerequisite course to Computer Science I, pushing them back one term from the traditional track students throughout the sequence.
These students often take Computer Science II in the Fall term.

Students who begin college with credit for Computer Science I take Computer Science II in the Fall. The students can gain credit for Computer Science I while still in high school through programs such as Advanced Placement Courses or Post-Secondary Education Option Programs. Students who achieve credit for Computer Science I while still in High School tend to have a high level of academic preparedness.

Due to the bimodal nature of Fall semester, the control group is composed of only Fall students from 2012 – 2014 (n = 205). The control group has three different instructors all of who have previous experience teaching this course. There are 63 students in the experimental group Fall 2015 term. The instructor for the experimental group does not have previous experience teaching this course. The results of this experiment failed to test significant in a two-tailed z-test with \( \alpha = .05 \) (\( p \)-value of .67%) with student progression slightly decreasing (73% to 70%). Figure 17 shows the final grade distribution for both groups.
The final experiment implementation for Computer Science II is in the Spring 2016 term. This group of students tends to be the most traditional group of students. Computer Science II Spring students tend to start the core sequence with Computer Science I when they arrive in the Fall, and, upon receiving a progressing grade (‘ABC’), enter Computer Science II in the Spring (See Figure 15). The control group for this experiment is comprised of Spring term students only from 2013 – 2015. A single experienced instructor teaches all sections for the control group. The experimental group also has a single experienced instructor (who is different from the control group instructor).

The progression rate shifts from 73% to 74%; this is not a statistically observable difference in a two tailed z-test with $\alpha = .05$. The resulting $p$ value is .77. Figure 18 charts the results:
the results. The shift in letter grades did not test significant in a two-tailed z-test with $\alpha = .05$ and a $p$-value of .18.

The students are binned according to their academic preparedness level upon entering college. These groupings are based on Math Placement Level scores or ACT-math scores (see Section 3.3.1). When the Computer Science II results are broken up by academic preparedness level, the Fall students are split between the two groups: high level of academic preparedness and low level of academic preparedness (See Figure 19). This is due to the bimodal nature of Computer Science II in the Fall as previously discussed. In this case, neither group has a statistically observable change in the new

---

**Figure 18 Computer Science II Spring Term Results**

This experiment is completed for Computer Science II students for Spring Term. These are the students on the traditional track. The slight increase in progression rate is not statistically observable.
pedagogy with barrier interventions pedagogy than the other groups.

Figure 19 Computer Science II Fall Term Results by Academic Preparedness
This figure illustrates the percentage of students who progressed forward through Computer Science II in the Fall semester. The hybrid traditional-active learning control group (HLA) and the fully flipped with barrier interventions (FI) groups are binned according to their academic preparedness level upon entering college.

When the Spring term students are binned by academic preparedness level, there is no statistically observable differences (See Figure 20).
4.2 Computer Organization

Computer Organization is the third course in the core sequence for Computer Science and Computer Engineering majors (See Figure 15). The course has a lower level focus than Computer Science I and II. It is the computer engineering course that introduces many hardware components to the students. Computer Organization has the following objectives:

- Perform calculations using binary, hexadecimal, and 2's complement number systems
- Interpret floating point numbers in IEEE-754 representation

Figure 20 Computer Science II Spring Term Results by Academic Preparedness

This figure illustrates the percentage of students who progressed forward through Computer Science II in the Spring semester. The hybrid traditional-active learning control group (HLA) and the fully flipped with barrier interventions (FI) groups are shown by the academic preparedness levels of the students upon entering college. Standard Error shown.
- Explain the function of the key components of a modern CPU, including the ALU, MUXs, buses, register files, and the memory interface
- Use the run-time stack to pass parameters between assembly language and high level code
- Explain the key concepts of polled, memory-mapped I/O, interrupt-driven I/O, and DMA
- Determine the effects of cache and memory hierarchy design decisions on overall memory performance

4.2.1 Methodology for Computer Organization

The methodology for this course is laid out similarly to the methodology for Computer Science I and II (see Sections 3.1 and 4.1.1). The control group is a hybrid traditional lecture-active learning environment described in Section 3.1.3. The experimental group is a group that uses intentional barrier interventions. The design is the same as in Section 3.1.5 except that only about half the lectures are given as videos. Also, a single experienced instructor is used for all Computer Organization experimental and control groups. The interventions used included:

- Classroom discussions on Computer Science and Engineering identity. One early in the term, and one late in the term.
- Students shown how to apply course material to career
- Midterm discussion and survey about how helpful the class is to students.
- Team activities that are the focus and driving force of classroom time.
• Frequent evaluations
• Video Lectures
• Classroom sessions recorded and available to students except for discussions where recording may have made students hesitant to share.

4.2.2 Computer Organization Experiment Results

4.2.2.1 Academic Term Progression Rates Different

Previously seen with other courses in the core sequence (Sections 3.3.1 and 4.1.2), Computer Organization has different demographic of students for different terms. Figure 21 illustrates that Fall term progresses is 82%, while Spring term progression is only 72%. This difference tested significant in a one tailed z-test with $\alpha = .05$. The $n$ values are 204 for Fall and 99 for Spring. The $p$-value is .046. Due to this difference, the control group is composed of only Fall students from 2012 - 2014.

![Figure 21 Computer Organization Different by Semester](image)

There is a significant difference between how many students progress through Computer Organization in the Spring term and how many progress in the Fall term.
4.2.2.2 Overall Progression Results

The intentional barrier interventions pedagogy is applied during the Fall 2015 term. This is when the traditional track students take the course. When the interventions are added to a Fall semester term the change from 82% to 85% is not statistically observable in a two tailed z-test with \( \alpha = .05 \) (\( n \) values of 150 and 54 for control and experimental groups respectively; \( p \) value of .58). Figure 22 reveals that the individual grade percentages have no notable differences as well.

![Pie charts showing progression rates](image)

*Figure 22 Computer Organization Results in Fall Term*
When the barrier interventions pedagogy is applied to Fall term of Computer Organization, very little change occurred.

4.2.2.3 Results Based on Different Academic Preparedness Levels

The results for the Computer Organization course experiment results for Fall 2015 term are binned by academic preparedness level in Figure 23. This results in \( n \) values for each
experimental group of 13, 15, and 10 for high, medium, and low preparedness levels respectively. There are no statistically observable differences.

![Bar Chart](image)

*Figure 23* Computer Organization Progression by Academic Preparedness
This figure illustrates the percentage of students who progressed forward through Computer Organization in the Fall semester. The hybrid traditional-active learning control group (HLA) and the fully flipped with barrier interventions (FI) groups are shown by the academic preparedness levels of the students upon entering college. Standard Error shown.

4.3 Data Structures and Algorithms

Data Structures and Algorithms is the final course in the core sequence (See Figure 15). After this course, students set their own paths. This is to allow the students flexibility to take the 4000 level electives that will give them the specific skills they want for the specific career path they wish to take. Of the students who successfully complete the Data Structures and Algorithms course with a letter grade of ‘D’ or better, 67.6% will graduate (this data is calculated on a 6-year bases and, therefore, uses the equivalent quarters course,
CS 400) (“Grad Rate by Grade Earned or Credit Range,” 2016). This course has the following objectives:

- Analyze basic algorithms for time and space complexity
- Design abstract data types appropriate for a given problem
- Implement data structures in an efficient manner
- Design and implement non-graphical user interfaces
- Select and implement appropriate data structures for a given problem
- Design algorithms to solve specific problems

4.3.1 Data Structures and Algorithms Methodology

The methodology for Data Structures and Algorithms is laid out identically to the methodology for Computer Organization (see Section 4.2.1) with two exceptions: labs and videos. While both courses have programming assignments, Computer Organization has an official lab time. The Data Structures and Algorithms course expects students to complete the programming assignments on their own time, seeking help as they need it. Both the control group and the experimental group for Data Structures and Algorithms have programming assignments without lab times.

The other difference only effects the experimental group (not the control group). In the Computer Organization experimental group, about half the lectures are provided as videos to be viewed before class. In the Data Structures and Algorithms course only about a quarter of the lectures are provided as video lectures before class. Even with more of the
lectures being delivered in class, the focus of the class remains on the activities. This reduction in video lectures prior to class does not result in a reduction of barrier interventions.

Other than these two changes, the experimental group is designed the same as the group described in Section 4.2.1, including the barrier interventions used. The control is the same hybrid traditional lecture-active learning environment described in previous experiments (see Section 4.1.1).

4.3.2 Data Structures and Algorithms Results

4.3.2.1 Data Structures and Algorithms Term Differences

The Data Structures and Algorithms course is different than the other courses in the core course sequence in that progression rates are not shown to be significantly different (in a one tailed z-test with $\alpha = .05$). The $p$-value is .12 for the increased progression from 63% to 69% in the Spring ($n = 191$) versus the Fall ($n = 197$) section (See Figure 24).

In Computer Science I and Computer Organization, the differences in progression rate by term are significantly different. Computer Science I has a 15 percentage points difference with $p = .0026$ (Section 3.3.1). Computer Organization has 10 percentage points difference with $p = .046$ (Section 4.2.2.1). Computer Science II does not have a significantly different term progression ($p = .48$), but it does have the bimodal distribution caused by students arriving at the University with credit for Computer Science I discussed in Section 4.1.2.

The loss of significantly different progression rates by terms may be caused by
decreasing importance of academic preparation upon enter college. As the students spend more time at the University, their preparation upon arrival may have a decreasing impact on their success. If that is the case, the results would show less difference in terms as courses progressed, as is witnessed here.

Even though the progression rates for the Data Structures and Algorithms course by term do not test significantly different, the terms are separated when looking at results. This is done for consistency with prior results.

![Figure 24 Data Structures & Algorithms Term Progression Differences](image)

The increase in number of students who progress in the Spring term versus the Fall term for Data Structures and Algorithms is not statistically observable. However, for consistency with the term-by-term analysis used for the other courses studied, the terms will continue to be separated for viewing the experimental results.

### 4.3.2.2 Data Structures and Algorithms Overall Results

The experiment for the Data Structures and Algorithms course is completed during the Spring 2016 term. This is when the traditional track students take the course. The control
group took the course in Spring terms between 2013 and 2015. It has two instructors. Both instructors taught for multiple terms that are included in the control, but one of those terms is the first term for one of the instructors. The experimental group also has two instructors. One of the instructors has previous experience teaching the course but has not taught it in four years. It is the first time the other instructor taught the course. The implementation of the interventions results in a non-significant increase in progression for this course: 69% to 70% with a $p$-value of .868 in a two tailed $z$-test. The $n$ values are 191 for control group and 105 for experimental group.

![Progression Rate](image)

**Figure 25 Data Structures & Algorithms Spring Results**
The flipped pedagogy with barrier interventions is applied to the Data Structures and Algorithms course during a Spring Term. This is when the traditional track students take the course. While progression slightly increase, the grade distribution of progressing students shifted unfavorably.

The grade distribution decrease seen in Figure 25 is not significant in a two tailed $z$-test
The nonsignificant change may have been caused by a less structured grading rubric. Higher level courses, such as Data Structures and Algorithms, tend to have fewer grading assistances. This makes them more susceptible to minor grade distribution changes based on the rigidness that the grader uses. Typically, there is a clearer concept of what is need to progress than what exact details distinguish ‘A’ work, ‘B’ work, and ‘C’ from each other. This can be compared to the Computer Science I and II series which typically has between 10 and 20 teaching/grading assistants. To keep consistency amongst all of the different graders, grading rubrics are distributed amongst the assistants. These rubrics include details for not only what to look for while grading but how many points should be removed for common errors.

4.3.2.3 Data Structures and Algorithms by Academic Preparedness

Once the students are binned according to their academic preparedness level the progression of the different groups is charted in Figure 26. There are no statistically observable differences.
Discussion of Individual Course Results

For any given course taken after Computer Science I, the changes are not statistically observable. While at a glance this might seem like the changes have little impact, it is important to recognize that the implementations are implemented sequentially. This sequential implementation with non-significant changes to progression rates may result in more students with lower levels of academic preparedness progressing through the program.
The pedagogy with barrier interventions is first implemented in Computer Science I in the Spring 2015 term. Sections 3.2.2, 3.3.2, and 3.4 discuss how the implementations of the new pedagogy progresses more students through the Computer Science I course than prior terms. It is expected that if more students are progressing through that course, the average academic preparedness level of students progressed will be lower. Therefore, when the Computer Science II experiment is implemented the following term (Summer 2015) the average academic preparedness level of the experimental students in the course should be lower than the average academic preparedness level in the control.

A similar result follows when looking at the Computer Organization experiment. Since the progression rate for Computer Science II course is similar or even slightly higher than the control, some of these less academic prepared students who progress through Computer Science I are progressing through Computer Science II to the Computer Organization course. When the Computer Organization experiment is implemented in Fall 2015, it contains additional students who progress due to the new pedagogy in Computer Science I. This reasoning can continue through the Data Structures and Algorithms course.

Therefore, even if the actual progression rate of students is not statistically different in the sequential courses examined in this chapter, the total number of students progressing forward could still be higher than it would have been without the addition of the barrier interventions.

4.5 Longitudinal Study
The impact of the interventions on Computer Science I courses resulted in an increased number of students that are eligible to enroll in Computer Science II. This progression is only beneficial though if the students continue to progress through Computer Science II and ultimately complete the entire program.

Ideally, students take the core courses one after another, moving forward one course each semester. Therefore, in this section of results, progression is defined as taking the next course in the sequential term of the prerequisite course. For example, a student who takes Computer Science II in the Fall must take Computer Organization in the Spring term in order to be given credit for being retained for the Computer Organization course. If the student took Computer Science II in the Fall, repeated Computer Science II in the Spring, and then takes Computer Organization in the Summer, that student would not be credited with progressing to the Computer Organization course in this study. The courses need to be sequential.

For this metric, progression from Spring term is demonstrated by enrollment in a subsequent course during the next term (Summer) or during the next term in the standard academic year (Fall). Thus, students who take Computer Science I in the Spring are given credit for taking Computer Science II in either the following Summer term or in the following Fall term. A student is considered to have progressed if they receive any final grade for the subsequent course (including failure or late withdrawal). (A student who signs up then drops the course is not considered retained).

4.5.1 Results

For this study, three groups are examined: Spring Computer Science I Students from
Section 3.2.2, Summer Computer Science I students from Section 3.3.2, and Fall Computer Science I students from Section 3.4.

To consider the impact on program progression, all the Computer Science and Computer Engineering students who took Computer Science I in the Spring term are selected for study. This results in 79 students for the control group (all Spring 2013 and 2014 sections in the hybrid lecture-active learning environment) and 29 students for the experimental group (Spring 2015 section that contains barrier interventions). Of these students selected for study the control has 39 students (49%) enroll in Computer Science II in the subsequent term; the experimental group has 19 such students (66%). Of these, 29 of the control group students (37%) take Computer Organization within two academic year terms of having taken Computer Science I. Likewise, 13 of the 29 students (45%) in the experimental group take Computer Organization two academic year terms. This increase in progression tests as significant for Computer Science II in a one-tailed t-test with $\alpha = .05$: $p = .0042$. It does not test as significant for Computer Organization: $p = .23$. Figure 27 shows the progression data of these students.
This analysis is repeated for the students who started Computer Science I in the Summer 2015 term. The control group contains 14 students from the Summer 2014 term. The experimental group contains 16 students with the barrier intervention pedagogy from the Summer 2015 term. Of those students, 8 control group students and 14 experimental students enroll in Computer Science II the following Fall. This 31 percentage point increase is statistically observable (one tailed t-test with $\alpha$ of .05 and a $p$ value of .034). The control group has only three students enroll in Computer Organization the following Spring. The
The experimental group has 11 students enrolled in Computer Organization the following Spring. This 48 percentage points increase is also significant with a $p$ value of just $0.0040\%$. Figure 28 shows the percentages of progression for the summer students. The 69% progression rate to Computer Organization for experimental students is greater than the percentage of progression for the control group to Computer Science II (only 57%).

**Figure 28** Percentage of students who enrolled in Computer Science II (1181) or both Computer Science II and Computer Organization (3310) in Sequential academic-year terms after completing Computer Science I in Summer 2015

This figure takes all the students who took Computer Science I in the Summer term and examines if they are retained for Computer Science II and Computer Organization in sequential terms. The control pedagogy’s $n = 14$ and the experimental pedagogy’s $n = 16$.

Finally, this process is completed for students who took Computer Science I in the Fall.
term. These would be the students on the traditional track for Computer Science and Computer Engineering students. The control group is composed of 399 students from Fall 2012 – 2014 terms. The experimental group that has the barrier interventions has 143 students from the Fall 2015 term. The number of the control students that enroll in Computer Science II the following Spring is 239; that is 60% of the control students that took Computer Science I in the Fall (See Figure 29). The number of the experimental group students that enroll in Computer Science II the following Spring is 87; that is 61% of the control students that took Computer Science I in the Fall. That difference is not significant in a one tailed z-test with α of .05 (p = .42). The control group has 176 students progress to Computer Organization; 44% of the original 399 Computer Science I students. The experimental group has 61 students progress to Computer Organization; 42% of the original 143 Computer Science I students. This difference is not significant either with a p value of .38.
4.5.2 Longitudinal Discussion

The purpose of changing the pedagogy from a hybrid lecture-active learning environment to a pedagogy that includes barrier interventions is to retain students that would have otherwise have left the program. At this time there is not enough information to comment on the entire program, but Section 4.5.1 looks at the progression of students through the Freshman level sequence (Computer Science I and II) and if they begin the
Sophomore level sequence (Computer Organization followed by Data Structures and Algorithms).

The non-traditional track terms have significant increases in progression rates for Computer Science II. The Summer 2015 students also has a significant increase for Computer Organization progression rates. This result holds with the discussion presented in Section 4.4: as more students progress through Computer Science I and the rates of progression does not change for sequential courses, the total number of students that progressed through increases.

The Spring 2015 students have a non-significant increase for Computer Organization. The progression rates are comparable to the fall progression rates. If this increase is significant, a sample size of 250 students is needed to observe it with statistical significance.

The Fall 2015 Computer Science I section does not have a significant increase in progression (Section 3.4). Therefore, as expected, there is not a significant difference in the number of students that progressed to Computer Science II or Computer Organization.

The Summer 2015 experimental group is the only group that has a statistically observable increase for progression into Computer Organization. The Spring 2015 students have a slight increase that may or may not have been caused by the interventions. Further research is needed in order to determine if the new pedagogy results in a consistent increase in progression through the program.

4.5.3 Retained Relevant Knowledge

While increases in progression rate speaks to the benefits of the new pedagogy, the
increase is only beneficial if the quality of students is not lowered. Section 3.3.3 introduces a system for assessing retained relevant knowledge and provides support that the students progressing from Computer Science I to Computer Science II are not doing so by lowering standards. Figure 30 shows the average quiz score for students enrolled in the Computer Organization course. There is not a significant difference in average quiz score.

![Figure 30 Retained relevant knowledge assessment for Computer Organization Spring Term](image)

This figure displays the average score on the retained relevant knowledge quiz taken at the beginning of Computer Organization.

### 4.6 Possible sources for error

Section 3.5 introduces some common sources of error that are relevant here as well. There are still errors due to human factors, different instructors, and data loss due to major changes. Additionally, the higher level classes generally have less detailed grading rubrics that could lead to more grade fluctuation as discussed in Section 4.3.2.2.
4.7 Student Survey on the New Pedagogy

The students are given a chance to give feedback on their view of different aspects of the pedagogy. Over the four courses, there are 129 participants in the survey. The survey asks the students to rate the impact on different interventions on each of the five barrier categories on a one (strongly disagree) to five (strongly agree) scale. (Not all 129 students answered every question. Questions pertaining to specific interventions not used in a course are removed). Table 4 summarizes the results of the survey.

<table>
<thead>
<tr>
<th>Key Elements of the Revised Course</th>
<th>INTEREST (My interest in CSE)</th>
<th>OPPORTUNITY (My confidence in ability to learn CSE content)</th>
<th>OPPORTUNITY (My belief in ability to succeed in CSE Career)</th>
<th>PSYCHOSOCIAL (My comfort with instructors and class participation)</th>
<th>PSYCHOSOCIAL (Feeling I am involved in actual CSE work)</th>
<th>ACADEMIC SKILLS /KNOWLEDGE (My learning and memory for course content)</th>
<th>COGNITIVE SKILLS (My ability to think critically / communicate concepts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity-focused Class Time</td>
<td>96</td>
<td>82</td>
<td>65</td>
<td>72</td>
<td>57</td>
<td>82</td>
<td>88</td>
</tr>
<tr>
<td>Frequent Quizzing</td>
<td>78</td>
<td>74</td>
<td>59</td>
<td>56</td>
<td>38</td>
<td>82</td>
<td>80</td>
</tr>
<tr>
<td>Open-ended Student Projects</td>
<td>63</td>
<td>62</td>
<td>63</td>
<td>47</td>
<td>65</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>Problem Solving Exercises</td>
<td>79</td>
<td>81</td>
<td>73</td>
<td>64</td>
<td>71</td>
<td>81</td>
<td>81</td>
</tr>
<tr>
<td>Recorded Lectures viewed Out of class</td>
<td>36</td>
<td>47</td>
<td>34</td>
<td>30</td>
<td>29</td>
<td>46</td>
<td>51</td>
</tr>
<tr>
<td>Group Centered Work</td>
<td>53</td>
<td>67</td>
<td>64</td>
<td>64</td>
<td>71</td>
<td>74</td>
<td>78</td>
</tr>
</tbody>
</table>

The Activity-focused Class Time, Problem Solving Exercises, and Group Centered Work interventions all increase all five success factors by more than 50 percent. Activity-focused Class Time and Problem Solving Exercises interventions receive a mean positive impact assessment of more than 75 percent. Dr Clark Shingledecker reported, “Overall
examination of the data showed that ratings indicating that a particular course element has produced a negative impact (greatly decreased or decreased) on any student success factor are uncommon (Mean 4.2 %).” (Appendix C – Retrospective Survey Findings).

Students report the Activity-focused Class Time intervention as the largest contributor to Interest, Opportunity (confidence in ability), Psychosocial (comfort interacting with others), and Cognitive Skills categories. It tied as the largest contributor for Academic Skills/Knowledge category along with the Frequent Quizzing intervention at 82 percent. The Problem Solving Exercises intervention immediately follows with 81 percent. Psychosocial (feeling involved in actual work) and Opportunity (belief in ability to succeed) categories are most impacted by the Problem Solving Exercises. Students report that the Group Centered Work interventions equally contributed to increase success in the Psychosocial (feeling involved in actual work) category.

According to the students’ surveys, the Recorded Lectures Viewed out of Class interventions are the least affective interventions in the survey. It consistently scores as the least impactful intervention for all seven categories surveyed.

4.8 Conclusion

In Chapter 3, more students are progressed through Computer Science I. This chapter explored the next three courses in the core sequence: Computer Science II, Computer Organization, and Data Structures and Algorithms. None of these courses have significant differences in the progression rates. With the progression rates remaining similar and the
number of students entering into the sequence increasing, there is a significant increase in the number of students progressing into Computer Science II. However, the increase into Computer Organization is only statistically observable for the Summer 2015 experimental group. Further research is needed to determine if there is a significant increase in progression rate.
Engineering and Computer Science is a field still largely dominated by white males. According to an American Communities Survey Report in 2011 women made up only 13% of the engineers and 27% of the computer related jobs (a percentage that has been decreasing since 1990) workforce (Landivar, 2013). The report continued, stating 70.8% of the STEM workforce is made up of Whites, not Hispanics or Latinos. Asians make up an additional 14.5% of the STEM workforce. This is more than 2.5 times their representation in the general workforce. African Americans, Hispanics, Latinos, and Native Americans have lower percentages of STEM workers than they have in the
general workforce. These subpopulations that do not fit the ‘typical’ STEM student are referred to as underrepresented groups (URG).

By their very nature, URG make up a smaller percentage of the Computer Science and Computer Engineering classroom. This may result in the impact of a pedagogy on the subgroup being masked by the impact on the more common student demographics. Therefore, in this chapter, special attention is given to explore the impact of the experiments described in Chapters 3 and 0 on the URGs of gender and ethnicity.

5.1 Underrepresented Group: Female students

5.1.1 Computer Science I

The results for the experiment of Computer Science I in Fall term is broken down by gender. This is the experiment for the traditional track Computer Science I students (see Section 3.4). The control group is the female students in the hybrid traditional lecture-active learning environment who took Computer Science I in the Fall 2012 – 2014 terms. The experimental group is the female students who took Computer Science I in Fall 2015. This group has the barrier interventions pedagogy. The control group of 46 female students has a progression rate of 72%. The experimental group of 17 female students has a progression rate of 82%. This did not test significant in a one tailed hypothesis t-test with $\alpha = .05$; the calculated $p$ value is .18. Figure 31 shows the results.
5.1.2 Computer Science II

The next experiment considers the effect of gender on the effectiveness of Computer Science II course interventions delivered during Fall terms. This experiment is separated by term to control for the impact of the bimodal distribution of progression due to initial term of study explored in Section 4.1.2. Female students in a hybrid lecture-active learning pedagogy in Fall terms between 2012 – 2014 comprise the control group. The experimental group consists of female students in the Fall 2015 term course offerings using a barrier intervention pedagogy. Once again a non-significant increase in female student progression is seen in a two-tailed hypothesis t-test with $\alpha = .05$; the calculated $p$ value is .54. Figure 32 shows the resulting 70% and 80% progression rates. In this case the control group only has 23 students and the experimental groups has 10.

![Pie charts showing progression rates](image)

*Figure 31 Computer Science I Progression Rates in Fall for Female students*

Female students had a nonsignificant increase to the progression rate when using a pedagogy with barrier interventions. The control group used a hybrid lecture-active learning pedagogy.
The next analysis considers the Computer Science II course for Spring students impact by gender. This experiment from Section 4.1.2 is composed of the traditional track students for Computer Science and Computer Engineering. The control group is composed of the female students in the hybrid lecture-active learning pedagogy in the Spring 2013-2015 terms. The experimental group is the female students in a barrier intervention pedagogy in the Spring 2016 term. In this case, the female student progression rate is a nonsignificant decrease (See Figure 33). The control group has 35 students and the experimental group has 13. The two tailed hypothesis t-test with $\alpha = .05$ results in a $p$ value of .74.

Figure 32 Computer Science II Progression Rates in Fall for Female students
Female students had a nonsignificant increase to the progression rate when using a pedagogy with barrier interventions. The control group used a hybrid lecture-active learning pedagogy.

The next analysis considers the Computer Science II course for Spring students impact by gender. This experiment from Section 4.1.2 is composed of the traditional track students for Computer Science and Computer Engineering. The control group is composed of the female students in the hybrid lecture-active learning pedagogy in the Spring 2013-2015 terms. The experimental group is the female students in a barrier intervention pedagogy in the Spring 2016 term. In this case, the female student progression rate is a nonsignificant decrease (See Figure 33). The control group has 35 students and the experimental group has 13. The two tailed hypothesis t-test with $\alpha = .05$ results in a $p$ value of .74.
5.1.3 Computer Organization

Computer Organization is the next course reviewed for gender-specific effects.

The female students from the Computer Organization experiment described in Section 4.2 are used for this analysis. This data is completed in the Fall 2015 term when the traditional track students take the course. The control group is the female students in a hybrid lecture active learning pedagogy from Fall 2012-2014 terms. The experimental group is the female students in the barrier intervention pedagogy in the Fall 2015 term. A slight decrease occurs in the progression rates of female students: 91% to 87% (see Figure 34). The decrease is not significant in a two-tailed t-test, with $n$ values of 11 and 8.
for the respective control and experimental groups and α of .05. The resulting $p$ value is .83.

![Graph showing data for control and experiment groups]

**Figure 34 Computer Organization Progression Rates in Fall for Female students**

Female students have a nonsignificant decrease to the progression rate when using a pedagogy with barrier interventions. The control group uses a hybrid lecture-active learning pedagogy.

5.1.4 Data Structures and Algorithms

The second course in the Sophomore-level sequence is Data Structures and Algorithms. The experiment in this course is explained in Section 4.3.2. It is completed in Spring term when the traditional track students take the course. The control group is the female students in the hybrid lecture-active learning pedagogy from Spring 2013 – 2015. The experiment group is the female students in the barrier interventions pedagogy from
Spring 2016. In this case, when the students are divided by gender, the percentage increased from 57% to 73% (see Figure 35). With 15 female students in the experimental group and 14 students in the control group, this does not test significant in a two tail t-test with $\alpha = .05$. The $p$-value is .38.

![Pie charts showing progression rates](image)

**Figure 35 Data Structures and Algorithms Progression Rate in Spring for Female students**

Female students had a nonsignificant increase to the progression rate when using a pedagogy with barrier interventions. The control group used a hybrid lecture-active learning pedagogy.

5.1.5 Hierarchical Linear Modeling for Female Students

Some of the individual course results are increases, but with the small $n$ values nothing is statistically observable. Hierarchical Linear Modeling (HLM) is a statistical method that allows for combining many small samples together in order to determine if
there is a general trend. It is a linear regression model that allows for offsets due to higher level details (i.e. which course the student is taking). Section 2.3.3 contains a complete explanation on HLM.

5.1.5.1 Data

The control group is female students who took one of the four core courses (Computer Science I, Computer Science II, Computer Organization, and Data Structures and Algorithms) in a hybrid lecture-active learning pedagogy from Fall 2012 through Fall 2015. The experimental group is female students who took one of the four core courses (Computer Science I, Computer Science II, Computer Organization, and Data Structures and Algorithms) in a barrier intervention pedagogy from Spring 2015 through Spring 2016. Table 5 gives a summary of the number of students.

<table>
<thead>
<tr>
<th></th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1180</strong></td>
<td>85</td>
</tr>
<tr>
<td>Control</td>
<td>63</td>
</tr>
<tr>
<td>Experimental</td>
<td>22</td>
</tr>
<tr>
<td><strong>1181</strong></td>
<td>88</td>
</tr>
<tr>
<td>Control</td>
<td>63</td>
</tr>
<tr>
<td>Experimental</td>
<td>25</td>
</tr>
<tr>
<td><strong>3100</strong></td>
<td>59</td>
</tr>
<tr>
<td>Control</td>
<td>44</td>
</tr>
<tr>
<td>Experimental</td>
<td>15</td>
</tr>
<tr>
<td><strong>3310</strong></td>
<td>31</td>
</tr>
<tr>
<td>Control</td>
<td>23</td>
</tr>
<tr>
<td>Experimental</td>
<td>8</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td>263</td>
</tr>
</tbody>
</table>
5.1.5.2 **Algorithm**

The algorithm runs a linear regression on each course to determine on a course by course bases. The regression is completed on the type of pedagogy used versus progression rate (Equation 1 in Section 2.3.3). After the course by course regression is complete, a linear regression runs on the course by course regressions allowing for an offset for each course (Equation 2 in Section 2.3.3).

5.1.5.3 **Script**

This analysis is performed using the R library lmer. The following code is executed:

```r
1. model = lmer(Progression ~ PedagogyGroup + (1|Course),
   data=myTable)
2. print(summary(model))
3. model.null = lmer(Progression ~ 1 + (1|Course),
   data=myTable, REML=FALSE)
4. model.alternative = lmer(Progression ~ PedagogyGroup +
   (1|Course), data=myTable, REML=FALSE)
5. print(anova(model.null, model.alternative))
```

The variable myTable holds the experimental and control group data. The first line of code creates the HLM for the data. The formula \(\text{Progression} \sim \text{PedagogyGroup} + (1|\text{Course})\) can be interpreted as using the pedagogy group (Control or Experimental) find the progression rate assuming a different base progression rate for each course.

Lines 3-5 are used to test the significance of the model. This is done by creating a null model in line three that disregards the pedagogy used. Then creating the alternative model in line four which matches the model given in line one, except that Residual Maximum Likelihood (REML) has been set to false. This needs to be set to false in order
to do the comparison using likelihood ratio tests. Line five runs an ANOVA likelihood ratio test comparing the two models created in lines three and four.

5.1.5.4 Results

The lmer function creates a model with the following information:

Linear mixed model fit by REML ['lmerMod']
Formula: Progression ~ PedagogyGroup + (1 | Course)
Data: myTable

Random effects:
Groups   Name        Variance Std.Dev.
Course   (Intercept) 0.0000   0.0000
Residual             0.2021   0.4496
Number of obs: 263, groups:  Course, 4

Fixed effects:                Estimate Std. Error    t value
(Intercept)                 0.69430    0.03236  21.455
PedagogyGroupExperimental  0.09141    0.06272   1.457

Since the random effects result for Course is zero, this means that for female students, the course has no noticeable effect on progression rates. The fixed effects table is interpreted as the control pedagogy (Intercept) has a progression rate of 69%. The experimental pedagogy (PedagogyGroupExperimental) has a progression rate that is 9 percentage points higher than control group (78%).

An ANOVA test is ran in order to determine the significance of this result. The ANOVA results show that the increased female progression rate determined by the HL model is not significant with a $p$-value of .14 in a Chi-square test.

5.1.6 Female Progression by Academic Preparedness

For this result the female students are binned according to their Academic Preparedness
Level. The specific course and term that the student are in is disregarded for this result. The control group is made up of female students with known academic preparedness level from all hybrid lecture-active learning pedagogies in each of the four core courses in the experiment from Fall 2012 through Fall 2015: Computer Science I, Computer Science II, Computer Organization, and Data Structures and Algorithms. The experimental group is all the female students from all the individual experiments of barrier intervention pedagogies from Spring 2015 through Spring 2016 with known academic preparedness levels. All tests are one-tailed t-tests with α of .05. The high level of academic preparedness students and the medium level of academic preparedness students does not have significantly different results: p-values of .46 and .082 respectively. The students with a low academic preparedness level has a significant increase from 51% to 76% (p = .025). Figure 36 shows the results of this study.
When results for the male students are binned according to their Academic Preparedness Level and the specific course, term that the student is in is disregarded, the males with a low level of academic preparedness did not have a statistically observable difference. The experimental group is all the male students from all the individual experiments of barrier intervention pedagogies from Spring 2015 through Spring 2016 with a low level of academic preparedness. The test is one-tailed t-tests with $\alpha$ of .05. The students with a low academic preparedness level does not have a statistically 

Figure 36 Female Progression Rates by Academic Preparedness Levels
Female students are binned according to their academic preparedness level upon entering college (High, Medium, or Low). The students with a low level of academic preparedness had a significant increase in their progression rate.
observable increase from 66.6% to 67.7% ($p = .417$, control $n = 314$, experimental $n = 96$).

5.2 Under-represented Ethnic Students

Just as female students are an URG within Computer Science and Computer Engineering, so are many different ethnicities. The two dominate ethnicities in Computer Science and Computer Engineering are White students (non-Hispanic, non-Latino) and Asian students. The other groups are considered under-represented groups (URG). This section looks at the effects of the experiments on this subgroup of students.

The number of total students in this category is smaller than the number of female students. Therefore, rather than separating the students out by course and term (which would be most ideal), each course is grouped together regardless of the term the student took the course.

5.2.1 Computer Science I

The underrepresented ethnicity students are pulled from the Computer Science I experiments described in Sections 3.2.2, 3.3.2 (Computer Science I Summer Term), and 3.4. The control group is composed of students who are an underrepresented ethnicity and took Computer Science I in a hybrid lecture-active learning pedagogy sometime between Fall 2012 and Fall 2014. The experimental group is composed of students who are an underrepresented ethnicity and took Computer Science I in a barrier intervention pedagogy between Spring 2015 and Fall 2015. The increase from 89% progression to 90% progression is not significant in a one tailed t-test with $\alpha$ of .05 and a $p$-value of .45 (see
Figure 37). The $n$ values are 44 control students and 10 experiment students.

![Pie charts showing progression rates](image)

**Figure 37 Computer Science I Progression Rates for Under-represented ethnic students**
Students from an underrepresented ethnicity had a nonsignificant increase to the progression rate when using a pedagogy with barrier interventions. The control group used a hybrid lecture-active learning pedagogy.

### 5.2.2 Computer Science II

The underrepresented ethnicity students are pulled from the Computer Science II experiments described in Section 4.1. The control group are composed of students who are an underrepresented ethnicity and took Computer Science II in a hybrid lecture-active learning pedagogy sometime between Fall 2012 and Spring 2015. The experimental group is composed of students who are an underrepresented ethnicity and took Computer Science II in a barrier intervention pedagogy between Summer 2015 and Spring 2016. The increase from 65% progression to 82% progression is not significant in a two tailed t-test with $\alpha$ of .05 and a $p$-value of .13 (see Figure 38). The $n$ values are 69 control students and 17
5.2.3 Computer Organization

The underrepresented ethnicity students are pulled from the Computer Organization experiments described in Section 4.2. The control group is composed of students who are an underrepresented ethnicity and took Computer Organization in a hybrid lecture-active learning pedagogy sometime between Fall 2012 and Summer 2015. The experimental group is composed of students who are an underrepresented ethnicity and took Computer Organization in a barrier intervention pedagogy Fall 2015. The decrease from 75% progression to 66% progression is not significant in a two-tailed t-test with $\alpha$ of .05 and a $p$-value of .73 (see Figure 39). The $n$ values are 20 control students and 6 experiment students.

Figure 38 Computer Science II Progression Rates for Under-represented ethnic students

Students from an underrepresented ethnicity had a nonsignificant increase to the progression rate when using a pedagogy with barrier interventions. The control group used a hybrid lecture-active learning pedagogy.
The decrease from 61% progression to 42% progression is not significant in a two-tailed t-test.

Figure 39 Computer Organization Progression Rates for Under-represented ethnic students

Students from an underrepresented ethnicity had a nonsignificant decrease to the progression rate when using a pedagogy with barrier interventions. The control group used a hybrid lecture-active learning pedagogy.

5.2.4 Data Structures and Algorithms

The under-represented ethnicity students are pulled from the Data Structures and Algorithms experiments described in Section 4.3. The control group is composed of students who are an under-represented ethnicity and took Data Structures and Algorithms in a hybrid lecture-active learning pedagogy sometime between Fall 2012 and Fall 2015. The experimental group is composed of students who are an under-represented ethnicity and took Data Structures and Algorithms in a barrier intervention pedagogy Spring 2016. The decrease from 61% progression to 42% progression is not significant in a two-tailed t-
test with \( \alpha \) of .05 and a \( p \)-value of .26 (see Figure 40). The \( n \) values are 46 control students and 14 experiment students.

5.2.5 Hierarchal Linear Modeling

Just as we saw with the female data, none of the individual courses test significantly different when looking at underrepresented ethnicity groups. Therefore, a Hierarchical Linear Model (HLM) is created in order to combine the courses (see Section 2.3.3 for a complete explanation on HLM).
5.2.5.1 Data

The control group is underrepresented ethnicity students who took one of the four core courses (Computer Science I, Computer Science II, Computer Organization, and Data Structures and Algorithms) in a hybrid lecture-active learning pedagogy from Fall 2012 through Fall 2015. The experimental group is underrepresented ethnicity students who took one of the four core courses (Computer Science I, Computer Science II, Computer Organization, and Data Structures and Algorithms) in a barrier intervention pedagogy from Spring 2015 through Spring 2016. Table 6 contains a summary of the number of students.

<table>
<thead>
<tr>
<th>Table 6 HLM Summary Data for Underrepresented ethnic students</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
</tr>
<tr>
<td>1180</td>
</tr>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Experimental</td>
</tr>
<tr>
<td>1181</td>
</tr>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Experimental</td>
</tr>
<tr>
<td>3100</td>
</tr>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Experimental</td>
</tr>
<tr>
<td>3310</td>
</tr>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Experimental</td>
</tr>
<tr>
<td>Grand Total</td>
</tr>
</tbody>
</table>

5.2.5.2 Script

This analysis is performed using the same script used in the HLM created for female students explained in Section 5.2.5.2. The myTable variable simply contains the
information on underrepresented students rather than female students.

5.2.5.3 Results

The lmer function creates a model including the following:

Linear mixed model fit by REML ['lmerMod']
Formula: Progression ~ PedagogyGroup + (1 | Course)
Data: myTable

Random effects:
Groups   Name        Variance Std.Dev.
Course   (Intercept) 0.01486  0.1219
Residual             0.19768  0.4446
Number of obs: 226, groups: Course, 4

Fixed effects:     Estimate Std. Error t value
(Intercept)       0.715795   0.070392   10.17
PedagogyGroupExperimental 0.001442 0.072941    0.02

The random effects portion shows that 1.5 percentage points of the progression rate variance is accounted for by the course. The fixed effects table states that the control pedagogy (Intercept) has a progression rate of 72%. The experimental pedagogy (PedagogyGroupExperimental) has a progression rate that is less than one percentage point higher than control group.

An ANOVA test run determines significance using a Chi-square test. The minimal change saw in the model is not statistically observable. The resulting $p$-value is .99.

5.2.6 Academic Preparedness Levels

Similar to the analysis in Section 5.1.6, the results for underrepresented ethnicity students the next analysis places students into bins based on their Academic Preparedness Level. The specific course and term that the student is in is disregarded for this result. The control group is made up of underrepresented ethnicity students with known academic
preparedness level from all hybrid lecture-active learning pedagogies in each of the four core courses in the experiment from Fall 2012 through Fall 2015: Computer Science I, Computer Science II, Computer Organization, and Data Structures and Algorithms. The experimental group is all the underrepresented ethnicity students from all the individual experiments of barrier intervention pedagogies from Spring 2015 through Spring 2016 with known academic preparedness levels. All tests are one tailed t-tests with $\alpha$ of .05. (Note that a significance test is not run for medium-level academic preparedness students since the experimental group consisted of only 3 students.) Figure 41 shows the results. The high level of academic preparedness students have a significant increase from 70% to 91% ($p = .033$). The students with a low academic preparedness level have a non-significant decrease from 66% to 63% ($p = .42$).
5.3 Overall Results of the Experiment

The experiments in this research is implemented over the Computer Science and Computer Engineering core sequence: Computer Science I, Computer Science II, Computer Organization, and Data Structures and Algorithms Course. Each course is evaluated individually in the previous chapters. Figure 42 charts the progression rates for the different courses for the different subgroups. The General category is all the students involved in the experiment. The other two categories are only female students and only
under-represented ethnic students.

5.4 Discussion

The effects of changes in the pedagogy on underrepresented groups can easily be masked by the impact on the larger groups as a whole. This chapter makes a specific point to look at the impact on some underrepresented populations.

5.4.1 Gender Discussion

The impact of the pedagogy with barrier interventions on female students is examined by course (Figure 31, Figure 32, Figure 33, and Figure 34). None of these show any significant difference in progression rates. A hierarchal linear model determines that the general trend found in the individual courses is also not significant (Section 5.1.5.4).

The results for female students are binned according to academic preparedness.
levels. This binning disregarded course, but the results of the hierarchal linear model found that course does not have an observable effect on progression rates for female students (Section 5.1.5.4). In this case, a significant increase is seen for students with a low level of academic preparedness (Figure 36). This result matches the results found in Computer Science I courses for the general population (Figure 9 and Figure 14). However, in the later sequence courses, this trend does not consistently hold true (Figure 19, Figure 20, Figure 23, and Figure 26). Figure 43 graphs the distribution of which courses the students with a low level of academic preparedness came from. It shows that Computer Science I students form more of the low academic preparedness level students than the other courses; however, it does not make up the overall majority. Therefore, the significant increase to progression rate for these students may be the result of the observed increases in progression for the general population in Computer Science I, or it may be because low academic preparedness female students continuously benefit from the barrier interventions pedagogy as they progress through the sequence.
5.4.2 Underrepresented Ethnicity

None of the results from the individual course analysis for underrepresented groups shows a significant difference between control and experimental groups (Figure 37, Figure 38, Figure 39, and Figure 40). There is also no general trend that is distinguishable when using hierarchal linear modeling (Section 5.2.5.3).

When the underrepresented ethnicity students are binned according to Academic Preparedness Level, the students entering with a high level of academic preparedness see a significant increase in progression (Figure 41). The low level of academic preparedness students do not see a significant change. This result is different than prior results (Figure 43).

*Figure 43 Distribution of Female Students with a Low Academic Preparedness Level by Course*

The female students had a significant increase in progression for students with low levels of academic preparedness when course is disregarded. The students from that test came from all four courses in the core sequence. Just under half the students came from a Computer Science I course.
9, Figure 14, Figure 19, Figure 20, Figure 23, Figure 26, and Figure 36). Generally, the high academic preparedness level students see only slight benefits to the interventions. This may speak to the need for non-cognitive barrier interventions for students from under-represented ethnicities. The lack of benefit to the low level of academic preparedness students may indicate that those students need a different set of barrier interventions.

5.5 Conclusion

The barrier intervention pedagogy increases the progression rate of the subpopulation of female students with a lower level of academic preparedness. This is consistent with the general population result seen in Computer Science I.

The subpopulation of under-represented ethnic students only sees a significant improvement in the case of students with a high level of academic preparedness. This change for the population that is most academically prepared may indicate that the students are previously failing to progress for non-cognitive reasons.
6 Retained Relevant Knowledge Assessment System

The Retained Relevant Knowledge Assessment System is a tool developed to provide a direct measurement of student performance throughout the program. This assessment system maps core curriculum courses’ learning objectives to IEEE-CS/ACM Computer Science and Computer Engineering knowledge topics recommended as program standards (The Joint Task Force on Computing Curricula - ACM/IEEE-Computer Society, 2013, 2016). The knowledge topics are then mapped to ABET student learning outcomes (ABET, 2016; Engineering Accreditation Commission, 2014).

A prerequisite quiz is developed to test incoming students on the knowledge topics that instructors assume the students have learned in prior courses. The results from all the course quizzes are collected and stored within a database to be used to assess the overall program effectiveness. The database also collects basic demographic information on the assessed students and data on final course grades. The database can then be used to produce useful reports, and monitor overall student performance in the entire program.
6.1 Motivation

6.1.1 Need for Continuous Assessment

ABET 2000 criteria requires engineering departments to adopt continuous program outcome assessment to satisfy basic level accreditation criteria. An effective assessment program is key to continuous improvement. Without a solid measure of student learning objectives, a cycle of improvement is driven by the variations and vagaries of the data and is less likely to result in meaningful positive change.

Historically, direct examinations such as the GRE, subject GRE, and Fundamentals of Engineering (FE) examination have been used to measure student educational achievement in University and to partially gauge professional competency. Examinations of this sort provide validation against a set of external criteria that demonstrate that the retained knowledge of each student is relevant to the current national standard. Unfortunately, end-of-program examinations of this sort make poor tools for continuous program improvement. It is difficult, if not impossible, to provide a linkage between overall examination performance and specific actions or pedagogies employed in the educational process that led to greater or lesser success.

Continuous periodic direct measurements provide the best opportunity for measuring the performance effects of specific changes to programs, courses, and pedagogies. However, such data collection efforts are practically limited due to the sometimes massive effort required from administration, faculty, and students.

6.1.2 Goals of this Assessment System
This infrastructure assesses program effectiveness with the following goals:

1. The assessment provides continuous periodic direct measurements of retained relevant knowledge.
2. The assessment outcome is immediately valuable to the assessment participants (students and faculty) as well as the continuous improvement of the program.
3. The assessment is not unduly burdensome.

6.2 Assessment Knowledge Topics

The goal of assessment is to provide data to measure (or illustrate a need for) improvement. The definition of the assessment standards then set a target goal towards which a program continuously strives to better meet. Although program objectives differ significantly among institutions, certain knowledge and skills are expected of graduates of engineering programs. We believe that the standard towards which programs should strive in Engineering is best communicated not only by the accreditation agencies but also by the appropriate discipline-specific international professional society. These societies maintain and regularly update the themes, knowledge areas, and professional practices expected of those entering their discipline.

For example, in computer science, the Joint Task Force on Computing Curricula between the Association for Computing Machinery (ACM) and IEEE-Computer Society provides regularly up-dated standards in curriculum, most recently in the volume Computer Science Curricula 2013 (CS2013) (The Joint Task Force on Computing Curricula - ACM/IEEE-Computer Society, 2013). The CS2013 Body of Knowledge organizes the expectations of
computing graduates into 18 Knowledge Areas (KA) which are created, revised, and removed as the discipline changes over time. Each of these KAs is further specified as a set of Knowledge Units each of which specifies a set of Knowledge Topics expected at the time of graduation.

<table>
<thead>
<tr>
<th>Table 7 CS2013 Knowledge Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
</tr>
<tr>
<td>AR</td>
</tr>
<tr>
<td>CN</td>
</tr>
<tr>
<td>DS</td>
</tr>
<tr>
<td>GV</td>
</tr>
<tr>
<td>HC</td>
</tr>
<tr>
<td>IAS</td>
</tr>
<tr>
<td>IM</td>
</tr>
<tr>
<td>IS</td>
</tr>
<tr>
<td>NC</td>
</tr>
<tr>
<td>OS</td>
</tr>
<tr>
<td>PD</td>
</tr>
<tr>
<td>PL</td>
</tr>
<tr>
<td>SDF</td>
</tr>
<tr>
<td>SE</td>
</tr>
<tr>
<td>SF</td>
</tr>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>
Table 8 Sample Knowledge Units in the Algorithms and Complexity Knowledge Area

<table>
<thead>
<tr>
<th>Algorithms and Complexity (AL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL/Basic Analysis</td>
</tr>
<tr>
<td>AL/Algorithmic Strategies</td>
</tr>
<tr>
<td>AL/Fundamental Data Structures and Algorithms</td>
</tr>
<tr>
<td>AL/Basic Automata Computability and Complexity</td>
</tr>
</tbody>
</table>

Table 9 Sample Knowledge Topics in the Algorithms and Complexity Knowledge Area: Fundamental Data Structures and Algorithms Knowledge Unit

- Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max, and mode in a list, approximating the square root of a number, or finding the greatest common divisor
- Sequential and binary search algorithms
- Worst case quadratic sorting algorithms (selection, insertion)
- Worst or average case O(N log N) sorting algorithms (quicksort, heapsort, mergesort)
- Hash tables, including strategies for avoiding and resolving collisions
- Binary search trees
- Common operations on binary search trees such as select min, max, insert, delete, iterate over tree
- Graphs and graph algorithms
- Representations of graphs (e.g., adjacency list, adjacency matrix)
- Depth- and breadth-first traversals
- Graphs and graph algorithms
- Shortest-path algorithms (Dijkstra’s and Floyd’s algorithms)
- Minimum spanning tree (Prim’s and Kruskal’s algorithms)
- Pattern matching and string/text algorithms (e.g., substring matching, regular expression matching, longest common subsequence algorithms)

For computer science programs, CS2013 can serve as a “gold standard” for contemporary computing education. The professional societies of other engineering disciplines provide similar international curricular standards along with, in many cases, examinations which new graduates are expected to pass in order to be fully qualified to work in the discipline.
In recognition that program objectives differ, CS2013 identifies topics as being either core tier-1 (required knowledge for every student in every program), core tier-2 (generally essential topic for which the vast majority should be covered but which may differ by student or program), or elective. CS2013 makes the categorizations by the process of “widespread consensus for inclusion” and further notes that “at least a preliminary treatment of most of these [core tier-1] topics typically comes in the first two years”. The explicitly stated coverage target for core tier-2 topics is “90-100% for every student, with 80% [as measured in lecture hours] considered as a minimum”.

While acknowledging that every program has differing educational objectives, use of professional society standards provides metrics which can gauge the success of the program against a national model. Such metrics suggest an infrastructure for direct assessment that allows comparison against discipline-wide expectations and to allow reflection on the need, causes, and appropriateness of any major deviations from the widespread consensus proposed by the discipline’s professional society.

6.3 METHODOLOGY

6.3.1 Mapping coverage

In an effort to compare this university’s students’ experiences to those across the nation, a mapping is created between university courses and the recommended knowledge topics from the professional societies: Computer Science Curricula (The Joint Task Force on Computing Curricula - ACM/IEEE-Computer Society, 2013) and Computer Engineering Curricula (The Joint Task Force on Computing Curricula - ACM/IEEE-Computer Society, 2013).
2016). Working with core program faculty, every mandatory course in Computer Science and Computer Engineering is mapped to the knowledge topics that the course requires prior to beginning the class (prerequisite knowledge) and to the knowledge topics developed within that course (see Figure 44).

<table>
<thead>
<tr>
<th>Algorithms and Analysis</th>
<th>CS BS Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differences among best, average, and worst case</td>
<td>x x p p</td>
</tr>
<tr>
<td>Asymptotic analysis of upper and average complexity bounds</td>
<td>x x p p</td>
</tr>
<tr>
<td>Big O notation: formal definitions</td>
<td>x p x p</td>
</tr>
<tr>
<td>Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential</td>
<td>x p x p</td>
</tr>
<tr>
<td>Empirical measurements of performance</td>
<td>x p x p</td>
</tr>
<tr>
<td>Time and space trade-offs in algorithms</td>
<td>x p x p</td>
</tr>
<tr>
<td>Big O notation: use</td>
<td>x p x p</td>
</tr>
<tr>
<td>Little o, big omega and big theta notation</td>
<td>x x</td>
</tr>
</tbody>
</table>

Figure 44 Mapping of Computer Science Core Courses to Knowledge Topics
Every core course in the Computer Science curriculum is mapped to the knowledge topics defined by the ACM/IEEE-CS joint task force. A 'p' indicates that the knowledge is expected as a prerequisite requirement for the course. An 'x' indicates that the knowledge topic is developed in the course. A 'px' indicates some prerequisite knowledge expected and that the knowledge will be expanded upon. This figure is a sample of the overall mapping completed for Computer Science.

The mapping is then extended to ABET criteria by mapping each knowledge topic to the ABET student learning outcome(s) (ABET, 2016; Engineering Accreditation Commission,
2014) covered (see Figure 45). This mapping between courses, knowledge topics, and student learning outcomes provides a way of looking at the overall development of student knowledge at different levels as students progress through the program. It also ensures that no gaps exist between what the instructor expects the students to know and what the students have already been taught.

<table>
<thead>
<tr>
<th>Algorithms and Complexity</th>
<th>Core Tier</th>
<th>Core I Hours</th>
<th>Core II Hours</th>
<th>CAC SLOs</th>
<th>CS BS Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms/Basic Analysis</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differences among best, average, and worst case</td>
<td>1</td>
<td>0.333</td>
<td>x</td>
<td>x</td>
<td>ab</td>
</tr>
<tr>
<td>Asymptotic analysis of upper and average complexity bounds</td>
<td>1</td>
<td>0.333</td>
<td>x</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>Big O notation: formal definition</td>
<td>1</td>
<td>0.888</td>
<td>x</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential</td>
<td>1</td>
<td>0.333</td>
<td>x</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>Empirical measurements of performance</td>
<td>1</td>
<td>0.888</td>
<td>x</td>
<td>ab</td>
<td></td>
</tr>
<tr>
<td>Time and space trade-offs in algorithms</td>
<td>1</td>
<td>0.333</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Big O notation: use</td>
<td>2</td>
<td>0.5</td>
<td>x</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>Little o, big omega and big theta notation</td>
<td>2</td>
<td>0.5</td>
<td>x</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 45 Mapping of Computer Science Knowledge Topics to ABET Student Learning Outcomes*

Every ACM/IEEE-CS knowledge topic is mapped to the ABET student learning outcomes (CAC SLOs a-k). Shown is a sample of this mapping.

6.3.2 Creating Direct Assessment Instruments

Using the mapping of knowledge topics to courses, a retained relevant knowledge quiz
is developed to test incoming students on the knowledge topics (and, by mapping, the associated ABET student learning outcomes) that instructors assume the students have learned in prior courses. To guarantee measures against national standards, an effort is made to use questions developed externally from the university (such as Computer Science subject GRE style questions) when possible. Instructors teaching subsequent courses (not the courses in which the topics are introduced) review and add questions as needed. Finally, the course coordinator and departmental curriculum committee review the questions to ensure that they are within the scope of the course. The distancing of the question creation from the common instructors is intentional. It is vital that the assessment of mastery be tied to national (and not instructor by instructor) based standards.

Consider the following segment of code in a java-like programming language. Assume that there are no syntax errors.

```java
int[] m = {2,3,4,5,6};
int n = 0;
int x = 0;
for (int val = 0; val < m.length; val++)
{
    if (val % 2 == 1)
    {
        n = n + val;
        x = x + 1;
    } // end-if
} // end-for
```

What is the most likely use for the code segment above?
A) Calculating the total sum of the values held in array m.
B) Calculating the average of the values held in array m.
C) Calculating the number of even values held in array m.
D) Calculating the average of odd values held in array m.
E) Calculating the number of values held in array m.

**Figure 46 Sample Retained Relevant Knowledge Question**
This sample question assesses the Knowledge Topic AL/Fundamental Data Structures: Numerical Algorithms. This topic is developed in Computer Science I and built upon in Computer Science II. Thus, this question would appear in the retained relevant knowledge assessment for Computer Science II.
The assessment is given to all students in the course at the beginning of the course as online quizzes using a standard Course Management System such as Blackboard or Desire2Learn. Staff members handle administrative details such as ensuring that the quizzes are posted for each core course. Whenever possible, the quiz is administered in an unused lab period at the beginning of the term. It is observed that this gives the fullest participation. If a lab period is not available, the quiz may be given during class, but it is more often given as a take home assignment. Most instructors chose to not have the quiz scores impact the students’ grades. This allows both more freedom for unsupervised administration and helps provide to the students a better low-impact measure of their preparedness for the course. Students are incentivized to take the perquisite quiz both by its results (indicating areas where they might need review or help from the course instructor) and sometimes additional instructor-based incentives (unlocking of online course materials, etc).

The quiz results help assess the retained knowledge of the students as they progress through the program without the bias of student’s opinions of their own knowledge (a common concern with indirect assessment measures). Currently, data is collected from nine courses that form the core of our computer science and computer engineering curricula.

6.3.3 Collecting the Assessment Results

Each term, the results from all the course quizzes are collected and stored within a database to be used to assess the overall program effectiveness. The assessment tools are delivered as online quizzes using a standard Course Management System such as Blackboard or Desire2Learn. An administrator is able to download the quiz results from
the online assessment tool and then uploads them directly into the database. This process has been semi-automated, reducing the required overhead. For the current nine courses, the downloading and uploading to the database process takes less than a half hour of faculty/staff effort per term.

The database also records basic demographic information on the assessed students. These questions are at the end of every student quiz and allow for more flexibility when trying to determine the impact of changes on various student subpopulations.

At the end of the term, administrators run an internal institutional report to collect data on final grades for the students. This report is added to the database. After this largely automated process, the database contains the relevant criteria for generating assessment reports. The details of the database are explained in Section 6.4.

6.3.4 Assessment Reports

The database is used to generate reports to aid in determining students’ development of knowledge throughout the program. The reports can summarize impact to help assess new pedagogies, changes in individual courses, subsections of the student population, and the overall curriculum. Details on how the database generates the reports are included in Section 6.4. Examples of the reports are located in Section 6.5.

6.4 Database

The data collected from the quizzes, mappings, and internal report (for final grades) is stored inside a database. The database is built using Ruby on Rails using a web interface. Currently the database is maintained on a local computer, but the web interfaces is
developed (see Figure 47). The intent is to ultimately deploy it as a website. The complete code for the database is in

Appendix B – Retained Relevant Knowledge Database Code.
Continuous Assessment System

Loading Data

Load Quiz Results
Load SLOs
Load Internal Report
Load SLOs Covered by Knowledge Topics

Standard Reports

Summary Report
Pedagogy Comparison
Prerequisite Quiz results from single section
ABET data collection summary course focus
ABET data collection summary slo focus
Quiz Coverage
Quizzes by Classes Report

Data Sheets

Answers
Class Average

Figure 47 The Database's Web Interface Homepage
From this homepage, a user can select to upload data into the database, generate standard reports, or view the table records.
6.4.1 Database Tables

The database tables can be broken into three parts: student focused tables, meta
evaluation tables, and summary tables. Figure 48 is the UML for the student focused tables.
There are five tables in this section: Students, Enrolls, ProgramOfStudys, Prerequisites,
and Answers. The Students table stores student demographics and basic college entry
statistics (i.e. ACT score). The student enrolls in a class. The Enrolls table stores
information pertinent to the moment in time when the course is taken (i.e. academic
progress) or to the course itself (i.e. grade in the course). While enrolled in that class the
student has one or more programs of study and has taken specific prerequisite courses to
get to qualify to take the current course. When the student begins a core course, the student
will take a retained relevant knowledge quiz and answer questions. The Answers table will
hold one record for every single question answered by the student. That record links to an
Enroll record and through that to the Student record.
Figure 48 UML of Student Focused Tables
The group of student focused tables contain information regarding students, the classes in which they enrolled, the prerequisite courses they took to enroll in a course, what program of study they are in at the time they enrolled in the course, and how they answered questions in the retained relevant knowledge assessment.

Figure 49 is the UML for the meta evaluation tables. (Note that the Answers table appears in both UMLs; it bridges the two subgroups of tables). The questions that the students answer are mapped to knowledge topics. A question may test more than one knowledge topic; the KtsCoveredByAnswers table contains the mapping of answered questions to knowledge topics. A knowledge topic may fulfill more than one student learning outcome; the SloCoveredByKts table contains the mapping between knowledge topics and student learning outcomes. More than one student learning outcome evaluates each program educational objective; PeoCoveredBySlos contains the mapping between
these two tables. The knowledge areas and knowledge units discussed in Section 6.2 are contained within the KnowledgeTopics table; each knowledge topic belongs to exactly one knowledge unit and knowledge area.

![Figure 49 UML of Meta Evaluation Tables](image)

The Answers table holds a record for every question answered by students in the retained relevant knowledge assessment. Those questions test knowledge topics. The knowledge topics cover student learning outcomes. The student learning outcomes fulfill program educational objectives. The Answers table is the same table that appears in the student focused tables, linking those tables to these meta evaluation tables.

This linkage through tables allows results to be analyzed many different ways. The student focused tables allow course by course, student by student analyses. The meta evaluation tables allow category analyses. With the Answers table linking the student focused tables to the meta evaluation tables, a combination of course information, student...
information, and meta categories is useable for analyses. The following style of questions can be answered:

- This term, across all courses, how well did students meet student learning objective CAC: a?
- How well did students in Computer Organization answer questions testing knowledge area Algorithms and Complexity Problems?
- Is there a significantly observable difference between students who take the prerequisite course (Introduction to Computer Programming) for Computer Science I and those that do not when looking at the results of the retained relevant knowledge quiz taken in Data Structures and Algorithms course?
- Is there an observable difference in Program Educational Objectives assessment based on ethnicity?

The final two tables in Figure 50 are summary tables. The data in these tables can be found by analyzing the data in the other tables, but for convenience the averages are stored. The SloAverages table stores the percentage of questions that are answered correctly and the number of questions answered for a Student Learning Outcome each term. The ClassAverages table stores the average quiz score and the number of students who took the quiz for a course in a specific term. Both these tables are used in the general summary report described in Section 6.4.4.
6.4.2 Uploading Results

There are two file types that are frequently uploaded to the database: retained relevant knowledge quiz results and internal institutional reports. Currently, ever term there are between five and fifteen quiz result files to upload. The exact number depends on the number courses, the number of sections, and whether sections used a common quiz distribution site (i.e. a common pilot website) that term. One internal institutional report is uploaded each term with final grades for the students.

In addition to the above standard files, occasionally files are uploaded for new student learning outcomes, new knowledge topics, etc. These files are addressed in Section 6.4.2.3.

6.4.2.1 Uploading Retained Relevant Knowledge Quiz Results

6.4.2.1.1 The Data File

Once students have taken the retained relevant knowledge quiz, an administrator
downloads the quiz results to a comma separated values file. Figure 51 shows a portion of the file. The actual file contains the following columns and content:

<table>
<thead>
<tr>
<th><strong>Column Header</strong></th>
<th><strong>Content</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Org Defined ID</td>
<td>The student’s wright state identification (w###xxx).</td>
</tr>
<tr>
<td>Username</td>
<td>The student’s system ID and email.</td>
</tr>
<tr>
<td>FirstName</td>
<td>The student’s first name.</td>
</tr>
<tr>
<td>LastName</td>
<td>The student’s last name.</td>
</tr>
<tr>
<td>Attempt #</td>
<td>A numerical value indicating how many times the student attempted the quiz (1 for first attempt).</td>
</tr>
<tr>
<td>Attempt Start</td>
<td>Time and Date stamp of when the student began the quiz.</td>
</tr>
<tr>
<td>Attempt End</td>
<td>Time and Date stamp of when the student submitted the quiz.</td>
</tr>
<tr>
<td>Section #</td>
<td>Numerical number determining the quiz section. (Quiz questions are divided into sections. These sections are determined by the quiz creator. Sections divide the questions by prerequisite concepts and general demographic questions.)</td>
</tr>
<tr>
<td>Q #</td>
<td>Numerical number identifying the question.</td>
</tr>
<tr>
<td>Q Type</td>
<td>Abbreviation to categorize the question style (i.e. MC for Multiple Choice).</td>
</tr>
<tr>
<td>Q Title</td>
<td>The question title provided by quiz creator but unseen to students. Question titles include knowledge topics tested by the quiz question.</td>
</tr>
<tr>
<td><strong>Q Text</strong></td>
<td>The actual question provided to students. This includes html markups that need to be removed.</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Bonus?</strong></td>
<td>A boolean value for marking a questions as bonus. (These quizzes do not have bonus questions so this value is always “FALSE”).</td>
</tr>
<tr>
<td><strong>Difficulty</strong></td>
<td>A numerical value to classify difficulty of questions. Currently this feature is unused; the default value of 1 appears in the file.</td>
</tr>
<tr>
<td><strong>Answer</strong></td>
<td>A possible answer to the question in cases where answer options are provided. This will be blank in other cases. (i.e. multiple choice questions will have a line in the file for every possible answer the student could have chosen. A fill in the blank question will only have one line in the file and this spot will be left blank.)</td>
</tr>
<tr>
<td><strong>Answer Match</strong></td>
<td>For fill-in questions, this box will contain the students answer. For questions with provided options, this box will contain “Checked” or “Unchecked” to indicate if a student selected the given answer.</td>
</tr>
<tr>
<td><strong>Score</strong></td>
<td>The number of points awarded for the quiz question.</td>
</tr>
<tr>
<td><strong>Out Of</strong></td>
<td>The number of possible points to earn for the quiz question.</td>
</tr>
</tbody>
</table>
Figure 51 A portion of a retained relevant knowledge quiz results file

This is a small portion of the retained relevant knowledge quiz results file. The Q Title column stores
information on which knowledge topics are tested by the question. Every possible answer to select
appears as its own row in the file. The Answer Matched column indicates whether the user selected
that particular answer for selection style questions.

125

Answer Match Score Out Of
Answer
Bonus? Difficulty
Q Text
Q # Q Type Q Title
Fundamental Programming Concepts
l
CS 1180 Com
higher-level
then
of ANY
1semantics and
equivalent)
and
1 (or
basic syntax
Programming
including
Checked
Intro to Computer
programming,
1 <p>CS1160
FALSE of computer
<p>This course requires students to be familiar with the fundemental concepts
Prerequisite course
1 MC
</div>l
of ANY higher-level
I (or equivalent)</p>
and1semantics
1 Science
syntax
Computer
basic
CS 1180
including
UnChecked
<p>Only
programming,
1 <div class="drt">
FALSE of computer
<p>This course requires students to be familiar with the fundemental concepts
Prerequisite course
1 MC
1 and1semantics of ANY higher-level l
including basic syntax
programming,UnChecked
1 <p>Other</p>
FALSE of computer
<p>This course requires students to be familiar with the fundemental concepts
Prerequisite course
1 MC
of ANY higher-level l
1semanticsabove</p>
1 andappearing
concepts
basic syntax
prerequisite
including
UnChecked
not familiar with
1 <p>I amprogramming,
FALSE of computer
<p>This course requires students to be familiar with the fundemental concepts
Prerequisite course
1 MC
1
1
2014
1
FALSE
<p>In what year did you complete the prerequisite course (or equivalent)?</p>
Year of prerequisite preparation
2 SA
1
1
UnChecked
1 <p>Fall</p>
FALSE
<p>In what term did you complete this prerequisite course (or its equivalent)?</p>
Term of prerequisite course
3 MC
1
1
UnChecked
1 <p>Spring</p>
FALSE
<p>In what term did you complete this prerequisite course (or its equivalent)?</p>
Term of prerequisite course
3 MC
1
1
Checked
1 <p>Summer</p>
FALSE
<p>In what term did you complete this prerequisite course (or its equivalent)?</p>
Term of prerequisite course
3 MC
1
1
UnChecked
1 <p>Other</p>
FALSE
<p>In what term did you complete this prerequisite course (or its equivalent)?</p>
Term of prerequisite course
3 MC
1 <p>If 1you took the prerequisite cours
Taylor(e.g. Smith)</p>
enter the 1last name of your instructor
FALSE
<p>If you took the prerequisite course at Wright State University, please
Instruction of prerequisite course
4 SA
/><br /><span>bool
100
100 errors.</span><br
there are no syntax
UnChecked
language. Assume that
1 <p>False</p>
FALSE
CAC a:EAC :DS/Basic Logic:Logical connectives <address><span>Consider the following segment of code in a java-like programming
6 MC
/><br /><span>bool
100
100 errors.</span><br
there are no syntax
Checked
language. Assume that
1 <p>True</p>
FALSE
CAC a:EAC :DS/Basic Logic:Logical connectives <address><span>Consider the following segment of code in a java-like programming
6 MC
/><br /><span>bool
100
100 errors.</span><br
there are no syntax
UnChecked
Assume that
depends</p>
language.
1 <p>It
FALSE
CAC a:EAC :DS/Basic Logic:Logical connectives <address><span>Consider the following segment of code in a java-like programming
6 MC
/><br /><span>bool
100
100 errors.</span><br
there are no syntax
UnChecked
language. Assume that
1 <p>Undefined</p>
FALSE
CAC a:EAC :DS/Basic Logic:Logical connectives <address><span>Consider the following segment of code in a java-like programming
6 MC
/><br /><span>bool
100
100 errors.</span><br
there are no syntax
UnChecked
that
know</p>
do notAssume
language.
1 <p>I
FALSE
CAC a:EAC :DS/Basic Logic:Logical connectives <address><span>Consider the following segment of code in a java-like programming
6 MC
<address>public class Ex
100
100
are no syntax errors.</span></p>
Checked
20</p>that there
Assume
1 <p>10
FALSE language.
the following segment of code in a java-like programming
call
<p><span>Consider
CAC a:EAC :PL/Object-Oriented Programming:Method
7 MC
<address>public class Ex
100
100
are no syntax errors.</span></p>
UnChecked
40</p>that there
Assume
1 <p>10
FALSE language.
the following segment of code in a java-like programming
call
<p><span>Consider
CAC a:EAC :PL/Object-Oriented Programming:Method
7 MC
<address>public class Ex
100
100
are no syntax errors.</span></p>
UnChecked
20</p>that there
Assume
1 <p>20
FALSE language.
the following segment of code in a java-like programming
call
<p><span>Consider
CAC a:EAC :PL/Object-Oriented Programming:Method
7 MC
<address>public class Ex
100
100
are no syntax errors.</span></p>
UnChecked
40</p>that there
Assume
1 <p>20
FALSE language.
the following segment of code in a java-like programming
call
<p><span>Consider
CAC a:EAC :PL/Object-Oriented Programming:Method
7 MC
<address>public class Ex
100
100
are no syntax errors.</span></p>
UnChecked
40</p>that there
Assume
1 <p>40
FALSE language.
the following segment of code in a java-like programming
call
<p><span>Consider
CAC a:EAC :PL/Object-Oriented Programming:Method
7 MC
100abstract class Dog {</address><
100
UnChecked <address>public
syntax errors.</p>
WOOF</p>
intentional
<p>WOOF
there are1no
FALSE
<p>Consider the following segment of java-like code. &nbsp;Assume that
CAC a:EAC :PL/Object-Oriented Programming:Subtyping
8 MC
100abstract class Dog {</address><
100
Checked <address>public
syntax errors.</p>
yip</p>
intentional
<p>WOOF
there are1no
FALSE
<p>Consider the following segment of java-like code. &nbsp;Assume that
CAC a:EAC :PL/Object-Oriented Programming:Subtyping
8 MC
100abstract class Dog {</address><
100
UnChecked <address>public
errors.</p>
yip</p>
syntax
WOOF
intentional
<p>WOOF
there are1no
FALSE
<p>Consider the following segment of java-like code. &nbsp;Assume that
CAC a:EAC :PL/Object-Oriented Programming:Subtyping
8 MC
100abstract class Dog {</address><
100
UnChecked <address>public
errors.</p>
WOOF</p>
yip syntax
intentional
<p>WOOF
there are1no
FALSE
<p>Consider the following segment of java-like code. &nbsp;Assume that
CAC a:EAC :PL/Object-Oriented Programming:Subtyping
8 MC
{</address><
method</p>
class Dog
is an abstract
100abstract
100
<address>public
because <em>speak()</em>
UnChecked
printederrors.</p>
is syntax
intentional
<p>Nothing
there are1no
FALSE
<p>Consider the following segment of java-like code. &nbsp;Assume that
CAC a:EAC :PL/Object-Oriented Programming:Subtyping
8 MC
that type potentially be assigned
100 of100
variable
or aless</p>
could
hundred
values
UnChecked
one
possible
1 <p>approximately
many unique/distinct
FALSE
<p><span>Consider a 32-bit data type such as java's integer data type. How
CAC a:EAC :PL/Basic Type Systems:Primative Types
9 MC
that type potentially be assigned
100 of100
could a variable
thousand</p>
values
UnChecked
one
possible
1 <p>approximately
many unique/distinct
FALSE
<p><span>Consider a 32-bit data type such as java's integer data type. How
CAC a:EAC :PL/Basic Type Systems:Primative Types
9 MC
that type potentially be assigned
100 of100
could a variable
million</p>
values
UnChecked
one
possible
1 <p>approximately
many unique/distinct
FALSE
<p><span>Consider a 32-bit data type such as java's integer data type. How
CAC a:EAC :PL/Basic Type Systems:Primative Types
9 MC
that type potentially be assigned
100 of100
could a variable
billion</p>
values
Checked
one
possible
1 <p>approximately
many unique/distinct
FALSE
<p><span>Consider a 32-bit data type such as java's integer data type. How
CAC a:EAC :PL/Basic Type Systems:Primative Types
9 MC
that type potentially be assigned
100 of100
a variable
or higher</p>
could
trillion
values
UnChecked
one
possible
1 <p>approximately
many unique/distinct
FALSE
<p><span>Consider a 32-bit data type such as java's integer data type. How
CAC a:EAC :PL/Basic Type Systems:Primative Types
9 MC


6.4.2.1.2 The Upload Experience

The data files are uploaded to the database system without any data cleaning through the web interface. Figure 47 shows the homepage for the web interface. The administrator clicks the “Load Quiz Results” option under the “Loading Data” section. This routes the administrator to the page seen in Figure 52. From here the administrator selects the file and fills in some basic information about the course. Once that is done, clicking the “Import” button will load the results into the database.
6.4.2.1.3 The Algorithm for Uploading the Quiz

The algorithm used to upload the results is in Table 10. The complete code is in the load_student_quiz_results view, controller, and model files in Appendix B.

Figure 52 The Database's Upload Quiz Results Webpage

To upload the retained relevant knowledge quiz results the user selects the csv file, enters information regarding the course, and clicks import. The database algorithm handles all cleaning of the data.
The clean and pull student data portion of the algorithm handles many of the issues with the data files. First, the data file contains many lines of data on quiz answers that are “Unchecked” by the student (see Figure 51). These lines are removed. Secondly, the system currently used occasionally downloads incomplete student results. In this case, if it cannot be determined if the student has already taken the course, the results are removed (this is covered in more detail later in this section). If it is known that this is the student’s first attempt at taking the course, the completed quiz sections are still used in the results. Finally, the current system often has blank lines in the files. These lines are removed.

Once an individual student’s results are selected from the results file, the algorithm checks to see if the student has previously taken the course. Some students retake courses due to dropping, withdrawing, or failing their first attempt. In these cases, students may have already seen the prerequisite quiz (as it does not always change from term to term). Also, since the quiz tests retained relevant knowledge, the previous attempt at the current course could improve the student’s ability to answer the quiz questions. Both these factors could result in an inflated grade on the quiz. As the goal of the quiz is to evaluate previous

---

Table 10 High Level Algorithm

<table>
<thead>
<tr>
<th>For each student who took the quiz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean and pull student data</td>
</tr>
<tr>
<td>If student has previously taken the course, skip student</td>
</tr>
<tr>
<td>Find or Create Student and Enroll records</td>
</tr>
<tr>
<td>Load quiz section questions (Technical or Demographics)</td>
</tr>
<tr>
<td>Update Enroll record to include quiz score</td>
</tr>
<tr>
<td>Adjust ClassAverage record to include this student</td>
</tr>
<tr>
<td>End For</td>
</tr>
</tbody>
</table>

---

128
course experiences, students who are retaking the course are removed from the results. However, if they have taken the retained relevant knowledge quiz on their first attempt at the course, that data will remain in the database unchanged.

The remainder of the algorithm handles the actual importing of the quiz results. A student record is found (or created) to link the student’s quiz results to previous obtained data. An Enroll record is created, recording the student is taking the current course. The Answer records will be linked to this Enroll record so the question can be linked to the term, year, and course in which it is answered. Next the sections of quiz questions are recorded. For demographic sections, the Student and Enroll records are updated with the latest demographic information. The technical questions are uploaded using the following algorithm:

**Table 11 Load Quiz Technical Questions Algorithm**

<table>
<thead>
<tr>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each question in this section</td>
</tr>
<tr>
<td>If Prerequisite option question</td>
</tr>
<tr>
<td>Load the prerequisite data</td>
</tr>
<tr>
<td>Else (It is a technical question)</td>
</tr>
<tr>
<td>Clean and find knowledge topics associated with the question</td>
</tr>
<tr>
<td>Remove HTML markup where appropriate</td>
</tr>
<tr>
<td>Load answer</td>
</tr>
<tr>
<td>Update number of questions answered and number questions answered correctly</td>
</tr>
<tr>
<td>End For</td>
</tr>
</tbody>
</table>

The technical quiz questions are broken into topic specific sections. Each section begins with questions regarding the prerequisite route followed (which university courses taken, which courses transferred from another university, etc.) that taught the knowledge
required to answer the questions in the current section. This data is stored in the Prerequisites table, and is linked to the Enroll record. This data can aid in determining if students in different paths actually gain and retain the same knowledge.

After the prerequisite questions are the technical questions. These are the results that actually determine the students retained relevant knowledge. Figure 46 is an example of a technical question. The knowledge topics associated with the questions are in the Q Title column. Some cleaning and searching is completed to locate all of the knowledge topics covered by the question.

Once all the knowledge topics are located, an Answer record is created. For each knowledge topic covered, a new record is added to the KtsCoveredByAnswers table linking each knowledge topic to the students answer record.

The total number of technical questions answered and the number of technical questions answered correctly is tracked. These are used in the final two steps of the High Level Algorithm (Table 10): Update Enroll record to include quiz score and Adjust ClassAverage record to include this student. The Enroll record includes the overall retained relevant knowledge quiz score. The ClassAverage record holds the average quiz score for all students taking the course that term.

6.4.2.2 Uploading Internal Institutional Report

The internal institutional report is a record containing information on students’ performance in classes as well as some basic background information. The relevant portion of the report is a table with the following columns:
<table>
<thead>
<tr>
<th>Column Header</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>University Identification Number (U############)</td>
</tr>
<tr>
<td>FULL_NAME_LFMI</td>
<td>Student’s full name. (Last, First M.)</td>
</tr>
<tr>
<td>GENDER</td>
<td>Student’s gender (M or F)</td>
</tr>
<tr>
<td>Campus Id</td>
<td>Campus Identification (w###xxx)</td>
</tr>
<tr>
<td>Programs</td>
<td>This is a multiple column field containing information on which programs the student has been enrolled in (i.e. Computer Engineering – Pre, Computer Science – BSCS, etc)</td>
</tr>
<tr>
<td>Courses</td>
<td>Next is a list of different courses. The column header states the course (i.e. CS 1150) and then the cell contains term year and grade (i.e. Fall 2015/C). If students has taken the course multiple times, multiple columns are used and the course title header is merged over the cells.</td>
</tr>
<tr>
<td>High School GPA</td>
<td>Student’s high school grade point average.</td>
</tr>
<tr>
<td>MPL</td>
<td>Student’s Math Placement Score.</td>
</tr>
<tr>
<td>ACT Math</td>
<td>Student’s ACT Math Component Score.</td>
</tr>
<tr>
<td>ACT Composite</td>
<td>Student’s ACT Composite Score.</td>
</tr>
</tbody>
</table>

To import the internal institutional report, select “Load Internal Report” under the Loading Data section from the database website homepage (see Figure 47). This will route
the administrator to the webpage in Figure 53. The file is selected and the “Load IR Report”
button is clicked. After that the database algorithm handles the import.

![Upload IR Report](image)

**Figure 53 The Database Upload Internal Institute Report Webpage**
An internal institute report is uploaded by selecting the file and clicking the Load IR report button.

### 6.4.2.2.1 Algorithm for Uploading the Internal Report

Table 12 contains the algorithm used for uploading the Internal Report. The complete
code is in the load_internal_reports view, controller, and model files in Appendix B.
The report has several header lines before the table with the content of interest. Since the table of interest has merged header lines for courses, a course identification table is created to associate a column with a specific course. Additionally, since the table shifts sizes depending on the number of major changes and course attempts, all variables are set dynamically by searching through the header options.

Once the initialization is complete, each student’s records are examined. Every student is one row in the csv file. Each course column is evaluated for the student. The cell may be empty and can be skipped. If the cell contains content, that content is divided to semester, year, and grade. If a student signed up for a course and then dropped it or if a
student is currently enrolled in the course, there is no grade provided. In this case the record is also skipped. If the student does have a grade, the associated Student record and course Enroll record grade is updated. (In case where there is already a grade in the Enroll record, the grade is overwritten. This is to reflect most current records in case of grade changes).

6.4.2.3 Uploading Other Files

The database currently has upload options for student learning outcomes and student learning outcomes covered by knowledge topics. These function simply by choosing the corresponding options from the upload section of the homepage.

The student learning outcomes upload file should be a comma separated values file which includes a header line. It should have the following four columns: accreditation body (“CAC”), title (“a”), description (“An ability to apply knowledge…”), and year added (2016).

The student learning outcomes covered by knowledge topics upload file should be a comma separated values file which includes a header line. It should have the following six columns: knowledge area (“SF”), knowledge unit (“Reliability through Redundancy”), knowledge topic (“Redundancy through…”), accreditation body (“CAC”), title (“a”), description (“An ability to apply knowledge…”), and year added (2016).

6.4.3 Viewing Database Entries

Many tables have links from the homepage under the Data Sheets section. Additionally, any table in the database can be viewed using the URL “<host>/tableName”. For example,
“http://localhost:3000/answers” displays the answers table.

6.4.4 Generating Reports

The data in the database can be combined in many different ways to generate many different reports. Section 6.5 explores some reports in regards of use. This section focuses on the programming behind the Summary Report. The Summary Report is designed to be an overview of all results. It is the report that is run each term after uploading the results by the retained relevant knowledge assessment. The report is a combination of the Every Course Report (Section 6.5.3.1), the Specific Course Report (Section 6.5.3.2), and the Across Program Report (Section 6.5.3.3). The exact code for generating this report is located in the standard_reports view, controller, and model.

Table 13 outlines the algorithm used to generate the summary report at a very high level. The first focus is on generating the Every Course Report (blue). Next the algorithm organizes the Across Program Report data (green). The algorithm ends be breaking down significantly different results in the prior two reports (orange).
6.4.4.1 **Every Course Report Algorithm**

The table ClassAverage stores the class averages. A simple query of the table for current semester and year will provide current class averages. However, in order to increase consistency in quiz deployment and accuracy of results, this method is not solely used. Instead a list of all courses that appear in the ClassAverage table are searched for individually. If a course that has previously stored data is not included in the database for the current term, it is given a value of 105% for graphing the current quiz score. A quick glance at the graph in this report will allow for a determination on what courses data is not collected. From there, the analysts can determine why that data is not present. Common reasons include the course not being offered, human error in failing to upload the data, or an instructor’s choice to not administer the quiz. Once the reason for the lack of information is determined, appropriate actions can be taken by administrators. Table 14 outlines this algorithm.

---

### Table 13 Summary Report Algorithm

<table>
<thead>
<tr>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Current Course Averages</td>
</tr>
<tr>
<td>Calculate Course Graphical Information</td>
</tr>
<tr>
<td>Create Course Summary Table</td>
</tr>
<tr>
<td>Get Student Learning Outcome Averages</td>
</tr>
<tr>
<td>Calculate Student Learning Outcome Graphical Information</td>
</tr>
<tr>
<td>Create Student Learning Outcome Summary Table</td>
</tr>
<tr>
<td>Breakdown Significantly Observable Differences for Course</td>
</tr>
<tr>
<td>Breakdown Significantly Observable Differences for Student Learning Outcomes</td>
</tr>
</tbody>
</table>
In calculating the course graphical information portion of the algorithm, the data needed to make the chart in the Every Course Summary Report is collected and organized. First a list of each unique course in created by querying the ClassAverage table. Then, for each course, the average quiz score except for the current term is queried from the ClassAverage table. In order to graph the past course averages, each average is converted into an (x, y) coordinates. The x value is an assigned value associated with the course. The y value is the average quiz score for that term of the course. Finally, a “Descriptive_Statistics” gem (Parkhurst, Brown, Farrell, Malmgren, & Egan, 2014) is used to generate the percentiles for a boxplot. Table 15 outlines the algorithm for this portion.

**Table 14 Get Current Course Averages Algorithm**

<table>
<thead>
<tr>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get a list of each unique course in ClassAverage table</td>
</tr>
<tr>
<td>For each unique course</td>
</tr>
<tr>
<td>If current term class average</td>
</tr>
<tr>
<td>Add current term class average to the list of current term averages</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>Add 105% to the list of current term averages</td>
</tr>
<tr>
<td>End for each unique class</td>
</tr>
</tbody>
</table>

Below the chart the report contains a table with some statistical information regarding the current term results (see X.X). The table contains the following columns and
<table>
<thead>
<tr>
<th>Column Header</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course</td>
<td>Identifies the course that the current row of data is about.</td>
</tr>
<tr>
<td>Current Number of Students</td>
<td>The number of students who contributed to the current terms average quiz score ($n$ value).</td>
</tr>
<tr>
<td>Current Quiz Average</td>
<td>The average score of the current quizzes taken this term by students who have no prior exposure to the course.</td>
</tr>
<tr>
<td>Number of Past Terms</td>
<td>Number of past terms that the retained relevant quiz has been administered.</td>
</tr>
<tr>
<td>Past Average</td>
<td>The average of the past average quiz scores.</td>
</tr>
<tr>
<td>Std Dev</td>
<td>The standard deviation of the current average quiz score when compared to past average quiz scores.</td>
</tr>
<tr>
<td>Significant (5%)</td>
<td>A Boolean value that is true if the p-value of the current quiz score is less than 5%; false otherwise.</td>
</tr>
</tbody>
</table>

Table 16 outline the algorithm used to create this table. The “Descriptive_Statistics” gem (Parkhurst et al., 2014) is used for calculating values.
### 6.4.4.2 Across Program Reports

To evaluate the program across all the courses, ABET student learning outcomes are used. Each quiz question is mapped to one or more knowledge topics. Then each knowledge topic is mapped to one or more student learning outcomes. This mapping from quiz questions to student learning outcomes allows for the same retain relevant knowledge quizzes to be used to assess students across the program.

Each knowledge topic may test more than one student learning outcome. In that case, the knowledge topic is included in each student learning outcome. Therefore, the quiz questions that correspond to that knowledge topic are included more than once in the result. Likewise, a quiz question that has multiple knowledge topics is included for each knowledge topic. The result is that each quiz question may be included in the calculations multiple times. However, an effort is made to limit the number of knowledge topics tested in a single question to one Computer Science knowledge topic and one Computer Engineering knowledge topic.

The first step in creating the Across Program Report is to update the SLOAverages table. Since the SLO average is calculated using all courses’ retained relevant knowledge quizzes,
it is not updated when individual courses are loaded. The Summary Report is designed to be run immediately upon completion of uploading all quiz results. Therefore, the SLOAverages table is updated as part of this algorithm. The questions answered in a given term that are mapped to knowledge topics that map to the desired SLO are used. The percentage of those questions answered correctly are logged in the SLOAverages table. Once the table has been updated, the current SLO averages are queried from the table. Table 17 outlines this portion of the algorithm.

**Table 17 Get Student Learning Outcome Averages Algorithm**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Determine most recent inclusion in SLO averages table</td>
</tr>
<tr>
<td>2</td>
<td>For each term that needs to be added to the SLO averages table</td>
</tr>
<tr>
<td>3</td>
<td>For each SLO</td>
</tr>
<tr>
<td>4</td>
<td>Using the questions answered in the given term that are mapped to knowledge topics that map to the given SLO calculate the percentage answered correctly</td>
</tr>
<tr>
<td>5</td>
<td>Load the data into the SLO Averages Table</td>
</tr>
<tr>
<td>6</td>
<td>End for each SLO</td>
</tr>
<tr>
<td>7</td>
<td>End for each term</td>
</tr>
<tr>
<td>8</td>
<td>Get the current SLOs from the SLO Averages Table</td>
</tr>
</tbody>
</table>

The next step in the algorithm queries the database for the student learning outcome information needed for the student learning outcomes summary chart in the report. For each student learning outcome which is assessed in the current term the past term student learning outcome averages are queried. The past scores are converted to (x, y) coordinates for graphing. The x value is the value assigned to that particular student learning outcome. The y value is the percentage of questions answered correctly that map to the specific student learning outcome. The percentiles are calculated using the Descriptive_Statistics
gem (Parkhurst et al., 2014). Table 18 outlines this portion of the algorithm.

**Table 18 Calculate Student Learning Outcome Graphical Information Algorithm**

<table>
<thead>
<tr>
<th>Create a list all SLOs for which data has been obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each SLO in list</td>
</tr>
<tr>
<td>Get past term averages for the SLO</td>
</tr>
<tr>
<td>Convert scores to points</td>
</tr>
<tr>
<td>Calculate Percentiles</td>
</tr>
<tr>
<td>End for each SLO</td>
</tr>
</tbody>
</table>

A summary table with statistical information regarding the current term student learning outcomes is included below the chart. The table contains the following columns and content:

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLO</td>
<td>Identifies the student learning outcome and accreditation body that the current row of data is about.</td>
</tr>
<tr>
<td>Current number</td>
<td>The number of questions that contributed to the current terms questions answered average quiz score (n value).</td>
</tr>
<tr>
<td>Current Average</td>
<td>The percentage of questions used out of the questions used.</td>
</tr>
<tr>
<td>Number Past Terms</td>
<td>Number of past terms that the SLO is assessed.</td>
</tr>
<tr>
<td>Past Average</td>
<td>The average of the past student learning outcome assessments.</td>
</tr>
<tr>
<td>Std Dev</td>
<td>The standard deviation of the current average when compared to past averages.</td>
</tr>
<tr>
<td>Significant (5%)</td>
<td>A Boolean value that is true if the p-value of the current average is less than 5%; false otherwise.</td>
</tr>
</tbody>
</table>
Table 19 outlines the algorithm used to create this table. The “Descriptive_Statistics” gem (Parkhurst et al., 2014) is used for calculating values.

Table 19 Create Student Learning Outcome Summary Table Algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>6.4.4.3 Breakdown Significantly Observable differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each SLO in list of current SLOs</td>
<td>When the difference for the term is significantly observable, the report drills down into the differences. For each course that is significantly different, that courses results are broken down by question. Since the query return two lines for each question (one row is count of answered correctly and one is a count of answered incorrectly), the query results are reformatted to make one summative line for the data. It includes the question, number of correct answers, total number who attempts, and the percentage that answered correctly.</td>
</tr>
<tr>
<td>Get past SLO averages</td>
<td></td>
</tr>
<tr>
<td>Calculate mean, standard deviation, significance</td>
<td></td>
</tr>
<tr>
<td>Compile data into table</td>
<td></td>
</tr>
<tr>
<td>End for each SLO</td>
<td></td>
</tr>
<tr>
<td>Get the courses that are significantly different</td>
<td></td>
</tr>
<tr>
<td>For each significantly different course</td>
<td></td>
</tr>
<tr>
<td>Query the count of number of correct and incorrect answers for each question</td>
<td></td>
</tr>
<tr>
<td>Reformat query into table output</td>
<td></td>
</tr>
<tr>
<td>End for each significantly different course</td>
<td></td>
</tr>
</tbody>
</table>

The report drills down the significantly observable differences for student learning
outcomes by four levels: knowledge areas, knowledge units, knowledge topics, and questions. Each level is queried for the number of correct and incorrect responses contributing to that category. Since the query returns two lines for each item (one row is count of answered correctly and one is a count of answered incorrectly), the query results are reformatted to make one summative line for the data. All the reformatting include a title column (Knowledge Area, Knowledge Unit, Knowledge Topic, and Question), number answered correctly, total number of answers, and percentage answered correctly. The Question section also included the knowledge areas, knowledge units, and knowledge topics associated with the question. Table 21 outlines this portion of the algorithm.

<table>
<thead>
<tr>
<th>Table 21 Breakdown Significantly Observable Differences for SLOs Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the SLOs that are significantly lower</td>
</tr>
<tr>
<td>For each significantly different SLO</td>
</tr>
<tr>
<td>Query the count of number of correct and incorrect answers</td>
</tr>
<tr>
<td>for each knowledge area that is mapped to the SLO</td>
</tr>
<tr>
<td>Reformat the knowledge area query into table output</td>
</tr>
<tr>
<td>Query the count of number of correct and incorrect answers</td>
</tr>
<tr>
<td>for each knowledge unit that is mapped to the SLO</td>
</tr>
<tr>
<td>Reformat the knowledge unit query into table output</td>
</tr>
<tr>
<td>Query the count of number of correct and incorrect answers</td>
</tr>
<tr>
<td>for each knowledge topic that is mapped to the SLO</td>
</tr>
<tr>
<td>Reformat the knowledge topic query into table output</td>
</tr>
<tr>
<td>Query the count of number of correct and incorrect answers</td>
</tr>
<tr>
<td>for each question that is mapped to the SLO</td>
</tr>
<tr>
<td>Reformat the question query into table output</td>
</tr>
<tr>
<td>End for each significantly different course</td>
</tr>
</tbody>
</table>

6.5 Results & Discussion

6.5.1 Results and Discussion of Continuous Direct Measurements of
Retained Relevant Knowledge

In this infrastructure, assessment quizzes for expected retained relevant knowledge are administered before every core course in the computer science and computer engineering curriculum (see Table 22). Most students will take roughly five quizzes over their first two years of the program. The course sequences and time lapse between the courses tends to be similar for lower level courses (generally taken in sequential terms with summer term optional).

<table>
<thead>
<tr>
<th>Course</th>
<th>Recommended Year Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science II</td>
<td>Freshman</td>
</tr>
<tr>
<td>OS Concepts and Design</td>
<td>Freshman</td>
</tr>
<tr>
<td>Digital Systems Design</td>
<td>Sophomore</td>
</tr>
<tr>
<td>Data Structures</td>
<td>Sophomore</td>
</tr>
<tr>
<td>Computer Organization</td>
<td>Sophomore</td>
</tr>
<tr>
<td>Software Engineering</td>
<td>Junior</td>
</tr>
<tr>
<td>Comparative Languages</td>
<td>Junior</td>
</tr>
<tr>
<td>OS Internals and Design</td>
<td>Junior</td>
</tr>
<tr>
<td>Micro-Processor Based Embedded Systems</td>
<td>Junior</td>
</tr>
</tbody>
</table>

Quizzes are administered to upperclassmen when they take required courses, but there is naturally more variance in when they choose to take these courses. Upperclassmen are intentionally given more flexibility in their schedule in order to take the elective courses that will best prepare them for their desired career. We do not currently perform
assessments in elective courses. Our final assessment (in Senior Design) is holistic and is not currently included in the reports automatically generated by the assessment infrastructure.

6.5.2 Results and Discussion of Immediate Value of Assessment to Participation

6.5.2.1 Students
When a course has a prerequisite course requirement, it may use the previous courses material in many different ways. It may directly build upon the previously earned knowledge. It may use only parts of the previous course knowledge. It may require the previous knowledge for completing current tasks, such as using statistics in assigned problems. It may just use the skills indirectly, such as expecting familiarity with following sequential processes. Often students are left to figure out on their own what exactly is required and how well they are prepared for the task. This can be particularly concerning for transfer students or students for whom significant time has passed since taking prerequisite coursework.

The retained relevant knowledge assessment instruments provide all students with the opportunity to directly see what is expected of them and provide feedback on how well each student is prepared for success in the course. A student can use the results of the retained relevant knowledge quiz to determine what topics they need to refresh or maybe even re-learn in order to be prepared for the course. The quiz score can also reinforce the
students’ views of how much they are learning throughout the program.

Upon completing the quiz, the students immediately see their scores as well as what questions they answered correctly and incorrectly. This is provided via the online assessment system used (Desire2Learn, 2016). Figure 54 shows an example of what is shown to the student when they finish the quiz.

![Question 16](image)

**Figure 54 An Example of a Student’s View of the Retained Relevant Knowledge Quiz Result**

Upon completing the retained relevant knowledge quiz, the students can review the results. This is an example of a question result provided by the Desire2Learn course management system used (Desire2Learn, 2016).

6.5.2.2 **Instructors**

The online assessment system gives an instructor the ability to look at all student results
This gives the instructor an accurate representation for how prepared the students are for the course: both strengths and weakness. This also allows the instructor to better determine if the students require a review of any prerequisite topics and how such time could be best used. The system also allows the instructor to see how individual student perform in comparison to current (or past) averages in order to gain insight into potential individual needs for particular students. Figure 55 illustrates an instructor-level view of the assessment quiz results.

![Figure 55 An Example Instructor View of the Results from the Retained Relevant Knowledge Assessment](image)

After students have taken the retained relevant knowledge assessment, the instructor can review the results through the course management site (Desire2Learn, 2016). The students’ responses to specific questions gives the instructor specific and direct feedback on strengths and weakness. The instructor can use this information when outline course material.

If there are any class-wide concerns, the instructor can address the problem in the current class and also alert the departmental curriculum committee and course coordinators of prerequisite courses to the concern. For example, one quiz question focuses on analysis of a code snippet that uses a queue. One term roughly half the students answered this question
correctly, using first in first out. But almost every other student answered using a stack structure, first in last out. The instructor of the prerequisite course is able to adjust his course to better meet the needs of his students on this topic. Meanwhile, the current instructor is able to immediately and directly correct the misconception when the topic is introduced again in his course.

6.5.3 Results and Discussion of Overall Program Assessment

6.5.3.1 Every Course Report

Each term, a report is generated to see how the current students’ retained relevant knowledge quiz scores compare to the previous terms quiz scores (Figure 56). This allows for longitudinal monitoring of the courses at a high level. If the quiz score is significantly lower than what is common for the course, a breakdown report is automatically generated for that course (Figure 57). The course can be broken down to look at knowledge areas, knowledge units, knowledge topics, or individual questions. This report is reviewed by the curriculum committee each term to determine if further action should be taken.
6.5.3.2 Specific Course Report

On occasion a specific course is looked at more closely. For example, a course may be

Figure 56 An automated Every Course Report

The Every Course Report is an automated report based on the results of the retained relevant knowledge assessments. The average quiz score is determined for each course that term. Any course that is statistically observable as different is noted in red in the summary table below. This reports allows for all courses to be monitored at a glance. Any obvious issues can be drilled down further.

A Value of 105 was assigned to courses that data wasn’t collected on this term
looked at more closely due to a change in pedagogy for the prerequisite course. This may also be done because of large changes in course content or changes in the overall course program. In this case a report may be generated in order to assess the impact of the change. Since this assessment is completed in the course that follows the course being completed, it largely removes potential instructor bias from the results. The report generated for this can be specialized to meet the needs of the assessment. Figure 57 gives an example of a very basic report that may be generated. In the future, automated reports for each instructor of a prerequisite course each term will be created, so the instructors have feedback on how their sections are doing compared to other sections of the same course.

**Figure 57 An automated Specific Course Report**

A specific course report allows for a drill down on potential issues in the program. This is an example of a Specific Course Report that is used when a course shows statistically observable differences in the Every Course Report.
6.5.3.3  Across Program Report

Each term an automated report is generated to access the entire program (Figure 58). Each quiz question, regardless of course, is used to determine students’ abilities to complete each of the ABET student learning outcomes. This allows for strengths and weaknesses across the entire program to be examined. With the Every Course Report (Section 5.4.1), a weakness in a specific ABET student learning outcome may be masked by strengths in other questions. This report is also reviewed by the curriculum committee each term. Similar to the Every Course Report, this report breaks down statistically observable differences. The student learning outcomes are broken down by knowledge areas, knowledge units, knowledge topics, and individual questions. This aids in locating possible problems in the program.
6.5.4 Results and Discussion of Assessment Burden

Most of the cost in the assessment structure comes in the early stages of constructing mappings, assessment questions, and the database. A mapping of knowledge topics to courses and student learning outcomes needs to be created. Appropriate quiz questions need to be written, vetted, and entered into a Course Management System quiz database.
A database system for uploading course grades, uploading quiz results, and automatically generating standard reports has to be deployed. This level of effort is consistent with the efforts associated with the construction of program self-studies for accreditation visits.

Once the initial cost of infrastructure development and deployment is complete, however, the term to term assessment process takes minimal overhead. Instructors merely need to instruct students to take the quiz. Administrative staff can ensure that the quizzes are available, uploading the quiz results into the assessment database, and provide the automated reports to the appropriate faculty and curriculum committees for review.

General maintenance is needed in maintaining an up-to-date mapping between knowledge topics, student learning outcomes, and courses. Some alterations to quizzes may be needed to reflect these changes.

6.6 Conclusion

The retained relevant knowledge assessment system provides a way to give continuous periodic direct measurements of retained relevant knowledge throughout a computer science or computer engineering curriculum. The direct assessment removes the bias and inconsistency that can occur with qualitative questions. The frequency and consistency of the measurements helps evaluators determine (and adjust if necessary) causal events.

The assessment gives immediate valuable feedback to students, faculty, and program reviewers. The students and instructors can better prepare for the course. The program reviewers can evaluate specific sections of the program, specific groups of students, or
specific skills if desired. The impact of intentional changes can be consistently monitored, and issues from unintentional changes may be spotted more promptly.

The online features that allow for the assessment quizzes to be given outside of class and the regular generation of automated reports minimizes the burden of completing the assessment. The low cost of the assessment makes the execution of the assessment every term reasonable.
7 Conclusion and Future Work

7.1 Implements Non-Cognitive Barrier Interventions within the Classroom

Non-cognitive issues can create barriers to student success. Previous attempts to address non-cognitive issues have been focused on additional programs such as extracurricular activities or first year seminar programs. In this experiment, interventions for some of these issues are addressed within the courses that the students already take. Students receive the benefits of the interventions without adding any additional time requirements to those already associated with their course load.

7.1.1 Future Work

The barriers categories in this experiment are implemented as a group of interventions. When the students are surveyed, they found the elements of activity-focused class time and problem solving exercises to be most impactful (Section 4.7). Further research should be completed to determine which interventions actually most impactful. A starting experiment could compare the impact of just the categories determined most beneficial from students in the survey and compare it to the current experiment. This would lend insight into how well the students identify what they need compared to what is provided. This research could
expand to explore if different subgroups (academic preparedness levels, under-represented group, etc) are impacted differently by different interventions and how many of different types of interventions are most ideal.

7.2 Increases in Progression Rates for Computer Science I Course

A flipped classroom with barrier interventions is implemented across a Computer Science and Computer Engineering core curriculum sequence. Some of the lectures are moved to pre-recorded videos and reading assignments that can be completed prior to class. Reducing the amount of lecture in the classroom freed up class time for other activities. Some of that additional classroom time is used to address common barriers to STEM student success: motivation and interest, opportunity, psychosocial skills, cognitive skills, and academic preparation.

A fully flipped classroom with barrier interventions, some of which addressed non-cognitive barriers, significantly increased progression rates in a first year Computer Science I course. The pedagogy most aids students with a low levels of academic preparedness. The increase in progression does not appear to have reduced the quality of student outcomes. The statistically observed increase in progression rate for the fully flipped with barrier interventions pedagogy allows traditionally less prepared students to perform at rates more in line with their more academically prepared counterparts. By contrast, no such increase is statistically observable when the classroom is fully flipped,
but did not contain the barrier interventions.

7.2.1 Future Work

The impact on the students on the standard track term for Computer Science I is not statistically observable even when analyzing the subgroup of students with a low level of academic preparedness when entering college (Section 3.4). Future research could explore the impact of the barrier interventions when applied in classes with different distributions of students.

7.3 Exploration of Pedagogical Impact over Sequential Courses

Most experiments on pedagogical research focus on a single course. In this experiment, the new pedagogy is implemented over a sequence of courses. The results show the impact of sequential courses implementing the new pedagogy.

7.3.1 Future Work

Computer Science I has increased progression rates. Sequential courses did not have statistically observable differences. Future work could explore if a pedagogy loses effectiveness as students become increasingly familiar with it.

7.4 Possible Increases in Student Progression Rate through Computer Science Core Curriculum

The flipped pedagogy with barrier interventions that is introduced in Computer Science
I is used in the next three subsequent core courses: Computer Science II, Computer Organization, and Data Structures and Algorithms. The additional (traditionally less prepared) students progressing through Computer Science I does not change the progression rates of the subsequent core courses in a statistically observable way. The interventions that aid in progression through the first course combined with no change in subsequent course progression rates results in a statistically observable increase in the number of students progressing through the sequence in some instances, but not all.

7.4.1 Future Work

Future work needs to include further monitoring of progression rates to determine if increases in progression rates are consistently greater. The current experiment implemented in the spring needs to increase the $n$ to 250 in order for the observed difference to be statistically observable. (The summer term experiment is already statistically observable).

The increases in progression rate are only beneficial if students actually finish the entire program. Future research should look at the impact on eventual graduation rates and time to graduation for students in the new pedagogy.

Currently, once the students enter a course with the interventions they remained in courses with barrier interventions. Future research could explore the marginal impact of each course in a chain of courses on student progression.

7.5 Increases in Progression Rate for Under-Represented Groups
By their very nature, under-represented groups (URG) make up a smaller percentage of the Computer Science and Computer Engineering classroom. This may result in the impact of a new pedagogy on the subgroup being masked by the impact on the more common student demographics. In an effort to increase diversity amongst Computer Scientists and Computer Engineers, special attention is given to the impact of the flipped pedagogy with barrier interventions on URG of ethnicity and gender.

7.5.1 Increases in Progression Rate for Female Students with a Low Level of Academic Preparedness

The barrier intervention pedagogy increased the progression rate of the subpopulation of female students with a lower level of academic preparedness. This is consistent with the general population result seen in Computer Science I.

7.5.2 Increases in Progression Rate for Under-Represented Ethnic Students with a High Level of Academic Preparedness

The subpopulation of under-represented ethnic students saw a significant improvement in the case of students with a high level of academic preparedness. This change to aiding the most academically prepared may indicate that the under-represented ethnic students are previously failing to progress for non-cognitive reasons.

7.5.3 Future Work

The observations for under-represented groups are not statistically observable when
viewed by course. The benefits seen for the low level of academic preparedness female students and the high level of academic preparedness under-represented ethnic students lends hope that the impact is positive and that the observations are not statistically observable due to low \( n \) values. The Spring and Summer track Computer Science I sections (which is impacted more than the standard track Computer Science I Fall students) only has three and two students respectively for female students and three and zero students respectively for under-represented ethnic students. Continued interventions in these courses to increase the number of students in these two sub groups could increase the statistically observability of any change due to the flipped pedagogy with barrier intervention on these sub groups of students.

Future work can expand upon the groups looked at to include the sub populations of students with disabilities and first generation students. (Neither group is explored in the initial results due to issues in obtaining institute data on these sub populations).

7.6 Retained Relevant Knowledge Assessment System

ABET 2000 criteria requires engineering departments to adopt continuous program outcome assessment to satisfy basic level accreditation criteria. An effective assessment program is key to continuous improvement. Without a solid measure of student learning objectives, a cycle of improvement is driven by the variations and vagaries of the data and is less likely to result in meaningful positive change. The Retained Relevant Knowledge Assessment System is a tool developed to provide a direct measurement of student
performance throughout the program.

7.6.1 A Student Knowledge Assessment that Maps Knowledge to a National Standard


7.6.2 A Student Knowledge Assessment that Continuously and Directly Measures Retained Relevant Knowledge

A prerequisite quiz is developed to test incoming students on the knowledge topics that instructors assume the students have learned in prior courses. The results from all the course quizzes are collected and stored within a database to be used to assess the overall program effectiveness. The database also collects basic demographic information on the assessed students and data on final course grades. The database can then be used to produce useful reports, and monitor overall student performance in the entire program.

The Retained Relevant Knowledge Assessment System provides a way to give continuous periodic direct measurements of retained relevant knowledge throughout a computer science or computer engineering curriculum. The direct assessment removes the bias and inconsistency that can occur with qualitative questions. The linking between
questions and nationally set knowledge topics, links results to a national standard. The frequency and consistency of the measurements helps evaluators determine (and adjust if necessary) causal events.

7.6.3 A Student Knowledge Assessment that is Valuable to All Participants and Program Evaluators

The assessment gives immediate valuable feedback to students, faculty, and program reviewers. The students and instructors can better prepare for the course. The program reviewers can evaluate specific sections of the program, specific groups of students, or specific skills if desired. The impact of intentional changes can be consistently monitored, and issues from unintentional changes may be spotted more promptly.

7.6.4 A Student Knowledge Assessment that is Not Unduly Burdensome

The online features that allow for the assessment quizzes to be given outside of class and the regular generation of automated reports minimizes the burden of completing the assessment. The low cost of the assessment makes the execution of the assessment every term reasonable.

7.6.5 Future Work

Future work with the database could implement a predictive analysis between the prerequisite quiz grade and the likeliness of a student receiving a passing grade in the
course. This could prevent students from moving forward in a program (and sometimes re-taking a single course 3 or more times) because they did not possess the needed foundation.

Students scoring poorly on the prerequisite quiz could begin receiving aid immediately. This could be done by expanding the tool to include links between specific quiz questions and review data. It could also be used to encourage enrollment in supplemental recitations (until the needed foundation is obtained).

Additional reports could be added to the semester reports. Currently general information in considered (results by course and by student learning outcomes). This could be expanded to include data on under-represented groups, how students in different pathways progress (transfer students, students who began in Computer Science I, students who took Introduction to Computer Programming as a prerequisite to Computer Science I), or how students with different levels of academic preparedness progress.

The database could be expanded to include additional details. For example, the database could grow to include some advising details; a mapping of knowledge topics desired for different common careers could be included. A student then could use the mapping between courses and knowledge topics to determine which courses would be most beneficial for a desired career.


http://doi.org/10.1073/pnas.1319030111


Grad Rate by Grade Earned or Credit Range. (2016). *Internal Report at Wright State University*. 167


http://doi.org/10.3389/fpsyg.2015.00415

STEM Attrition: College Students’ Paths Into and Out of STEM Fields. (2014).

*Statistical Anaylsis Report.*


*Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* Practice. http://doi.org/10.1145/2534860


reporting research on fist-year seminars, volume II (Monograph No. 42). *Columbia, SC: University of South Carolina, National Resource Center for The First-Year Experience and Students in Transition.*


Appendix A – Validation Methods

Well-Established Tools

In cases of well-established tools, the first validation step verifies an understanding of the tool. A case with a known solution is used. The data for the case is ran through the tool to ensure the same result is produced. After an understanding of the tool is established, the tool is used. While in use, verification of known statistics (such as count and mean) check data upload into the tool.

Created Tools

Created tools went through a testing process. As portions of the code are made, select cases (at least one for each path through the code) are selected. For these test cases the code is traced step by step to ensure the desired result.

When able, the results are verified by an external system. For example, the class average quiz scores are verified via the Course Management System quiz scores.
Appendix B – Retained Relevant Knowledge Database Code

The Retained Relevant Knowledge Database Code is written in Ruby on Rails. Ruby on Rails contains a large amount of auto generated code. For example, when a database table is created, code is auto-generated to view the table, create an entry, edit an entry, update an entry, and remove an entry. Some of these auto-generated files have slight alterations in order to improve the user interface (i.e. reordering columns in view). Most of the code written for this project is located in load files and the standard reports files.
File
D:\railsApps\playgroundJan2016\Gemfile
1 source 'https://rubygems.org'
2
3 # Bundle edge Rails instead:
gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.1'
# Use mysql as the database for Active Record
gem 'mysql2', '~> 0.3.18'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .coffee assets and views
gem 'coffee-rails', '~> 4.1.0'
# See https://github.com/rails/execjs#readme for more supported runtimes
  gem 'therubyracer', platforms: :ruby
16 # Use jquery as the JavaScript library
  gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read more: https://github.com/rails/turbolinks
  gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
  gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc
25 # Use ActiveModel has_secure_password
  gem 'bcrypt', '~> 3.1.7'
28 # Use Unicorn as the app server
  gem 'unicorn'
31 # Use Capistrano for deployment
    gem 'capistrano-rails', group: :development
34 group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get a debugger console
    gem 'byebug'
37 # Access an IRB console on exception pages or by using <%= console %> in views
    gem 'web-console', '~> 2.0'
42 # Windows does not include zoneinfo files, so bundle the tzinfo-data gem
   gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
45 #Gems added for completing importing data
  gem 'roo'
  gem 'spreadsheet'
  gem 'chartkick'
  gem 'highcharts-rails'
gem 'descriptive_statistics', '~> 2.4.0', :require => 'descriptive_statistics/safe'

GEM
remote: https://rubygems.org/
specs:
  actionmailer (4.2.1)
  actionpack (= 4.2.1)
  activejob (= 4.2.1) mail
  (~> 2.5, >= 2.5.4) rails-dom-testing (~> 1.0, >= 1.0.5)
  actionpack (4.2.1)
  actionview (= 4.2.1)
  activesupport (= 4.2.1)
  rack (~> 1.6) rack-test
  (~> 0.6.2) rails-dom-testing (~> 1.0, >= 1.0.5)
  rails-html-sanitizer (~> 1.0, >= 1.0.1)
  actionview (4.2.1)
  activesupport (= 4.2.1)
  builder (~> 3.1)
  erb (~> 2.7.0)
  rails-dom-testing (~> 1.0, >= 1.0.5)
  rails-html-sanitizer (~> 1.0, >= 1.0.1)
  activejob (4.2.1)
  activesupport (= 4.2.1)
  globalid (>= 0.3.0)
  activemodel (4.2.1)
  activerecord (= 4.2.1)
  activesupport (= 4.2.1)
  builder (~> 3.1)
  coffeescript-source execjs
  coffeescript-source (1.10.0)
  concurrent-ruby (1.0.0)
  descriptive_statistics (2.4.0)
  erubis (2.7.0) execjs (2.6.0)
  globalid (0.3.6)
  activesupport (~> 4.1.0)
  highcharts-rails (4.1.9)
  railties (~> 3.1)
  i18n (0.7.0)
  jbuilder (2.4.0)
  activesupport (~> 3.0.0, < 5.1)
  multi_json (~> 1.2)
  jquery-rails (4.0.5)
  rails-dom-testing (~> 1.0)
  railties (~> 4.2.0)
  thor (~> 0.14, < 2.0)
  json (1.8.3)
  loofah (2.0.3)
  nokogiri (~> 1.5.9)
  mail (2.6.3)
  mime-types (~> 1.16, < 3)
  mime-types (2.99)
  mini_portile2 (2.0.0)
  minitest (5.8.3)
  multi_json (1.11.2)
  mysql2 (0.3.20-x64 mingw32)
  nokogiri (1.6.7.1-x64 mingw32)
  mini_portile2 (~> 2.0.0.rc2)
  rack (1.6.4)
  rack-test (0.6.3)
  rack (~> 1.0)
  rails (4.2.1)
  actionmailer (~ 4.2.1)
  actionpack (~ 4.2.1)
  actionview (~ 4.2.1)
  activejob (~ 4.2.1)
  activemodel (~ 4.2.1)
  activerecord (~ 4.2.1)
  activesupport (~ 4.2.1)
  builder (~ 3.1)
  activerecord (4.2.1)
  activemodel (~ 4.2.1)
  activesupport (~ 4.2.1)
  railties (~> 1.0)
  coffeescript-source
  coffeescript-source (1.10.0)
  concurrent-ruby (1.0.0)
  descriptive_statistics (2.4.0)
  erubis (2.7.0) execjs (2.6.0)
  globalid (0.3.6)
  activesupport (~> 4.1.0)
  highcharts-rails (4.1.9)
  railties (~> 3.1)
  i18n (0.7.0)
  jbuilder (2.4.0)
  activesupport (~> 3.0.0, < 5.1)
  multi_json (~> 1.2)
  jquery-rails (4.0.5)
  rails-dom-testing (~> 1.0)
  railties (~> 4.2.0)
  thor (~> 0.14, < 2.0)
  json (1.8.3)
  loofah (2.0.3)
  nokogiri (~> 1.5.9)
  mail (2.6.3)
  mime-types (~> 1.16, < 3)
  mime-types (2.99)
  mini_portile2 (2.0.0)
  minitest (5.8.3)
  multi_json (1.11.2)
  mysql2 (0.3.20-x64 mingw32)
  nokogiri (1.6.7.1-x64 mingw32)
  mini_portile2 (~> 2.0.0.rc2)
  rack (1.6.4)
  rack-test (0.6.3)
  rack (~> 1.0)
  rails (4.2.1)
  actionmailer (~ 4.2.1)
  actionpack (~ 4.2.1)
  actionview (~ 4.2.1)
  activejob (~ 4.2.1)
  activemodel (~ 4.2.1)
  activerecord (~ 4.2.1)
  activesupport (~ 4.2.1)
  builder (~ 3.1)
  coffeescript-source
  coffeescript-source (1.10.0)
  concurrent-ruby (1.0.0)
  descriptive_statistics (2.4.0)
  erubis (2.7.0) execjs (2.6.0)
  globalid (0.3.6)
  activesupport (~> 4.1.0)
  highcharts-rails (4.1.9)
  railties (~> 3.1)
  i18n (0.7.0)
  jbuilder (2.4.0)
  activesupport (~> 3.0.0, < 5.1)
  multi_json (~> 1.2)
  jquery-rails (4.0.5)
  rails-dom-testing (~> 1.0)
  railties (~> 4.2.0)
  thor (~> 0.14, < 2.0)
  json (1.8.3)
  loofah (2.0.3)
  nokogiri (~> 1.5.9)
  mail (2.6.3)
  mime-types (~> 1.16, < 3)
  mime-types (2.99)
  mini_portile2 (2.0.0)
  minitest (5.8.3)
  multi_json (1.11.2)
  mysql2 (0.3.20-x64 mingw32)
  nokogiri (1.6.7.1-x64 mingw32)
  mini_portile2 (~> 2.0.0.rc2)
  rack (1.6.4)
  rack-test (0.6.3)
  rack (~> 1.0)
  rails (4.2.1)
<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>rails-dom-testing</td>
<td>(1.0.7)</td>
</tr>
<tr>
<td>active-support</td>
<td>(&gt;= 4.2.0.beta, &lt; 5.0)</td>
</tr>
<tr>
<td>nokogiri</td>
<td>(~&gt; 1.6.0)</td>
</tr>
<tr>
<td>rails-deprecated-sanitizer</td>
<td>(&gt;= 1.0.1)</td>
</tr>
<tr>
<td>rails-html-sanitizer</td>
<td>(1.0.2)</td>
</tr>
<tr>
<td>loofah</td>
<td>(~&gt; 2.0)</td>
</tr>
<tr>
<td>raities</td>
<td>(4.2.1)</td>
</tr>
<tr>
<td>actionpack</td>
<td>(= 4.2.1)</td>
</tr>
<tr>
<td>active-support</td>
<td>(= 4.2.1)</td>
</tr>
<tr>
<td>rake</td>
<td>(&gt;= 0.8.7)</td>
</tr>
<tr>
<td>thor</td>
<td>(&gt;= 0.18.1, &lt; 2.0)</td>
</tr>
<tr>
<td>rake</td>
<td>(10.4.2)</td>
</tr>
<tr>
<td>rdoc</td>
<td>(4.2.1)</td>
</tr>
<tr>
<td>json</td>
<td>(~&gt; 1.4)</td>
</tr>
<tr>
<td>nokogiri</td>
<td>(~&gt; 1)</td>
</tr>
<tr>
<td>rubyzip</td>
<td>(~&gt; 1.1, &lt; 2.0.0)</td>
</tr>
<tr>
<td>ruby-ole</td>
<td>(1.2.12)</td>
</tr>
<tr>
<td>rubyzip</td>
<td>(1.1.7)</td>
</tr>
<tr>
<td>sass</td>
<td>(3.4.20)</td>
</tr>
<tr>
<td>sass-rails</td>
<td>(5.0.4)</td>
</tr>
<tr>
<td>raities</td>
<td>(&gt;= 4.0.0, &lt; 5.0)</td>
</tr>
<tr>
<td>sass</td>
<td>(~&gt; 3.1)</td>
</tr>
<tr>
<td>sprockets</td>
<td>(&gt;= 2.8, &lt; 4.0)</td>
</tr>
<tr>
<td>sprockets-rails</td>
<td>(&gt;= 2.0, &lt; 4.0)</td>
</tr>
<tr>
<td>tilt</td>
<td>(&gt;= 1.1, &lt; 3)</td>
</tr>
<tr>
<td>sdoc</td>
<td>(0.4.1)</td>
</tr>
<tr>
<td>json</td>
<td>(~&gt; 1.7, &gt;= 1.7.7)</td>
</tr>
<tr>
<td>rdoc</td>
<td>(~&gt; 4.0)</td>
</tr>
<tr>
<td>spreadsheet</td>
<td>(1.1.0)</td>
</tr>
<tr>
<td>ruby-ole</td>
<td>(&gt;= 1.0)</td>
</tr>
<tr>
<td>sprockets</td>
<td>(3.5.2)</td>
</tr>
<tr>
<td>concurrent-ruby</td>
<td>(~&gt; 1.0)</td>
</tr>
<tr>
<td>rack</td>
<td>(&gt; 1, &lt; 3)</td>
</tr>
<tr>
<td>sprockets-rails</td>
<td>(3.0.0)</td>
</tr>
<tr>
<td>actionpack</td>
<td>(&gt;= 4.0)</td>
</tr>
<tr>
<td>active-support</td>
<td>(&gt;= 4.0)</td>
</tr>
<tr>
<td>sprockets</td>
<td>(&gt;= 3.0.0)</td>
</tr>
<tr>
<td>thor</td>
<td>(0.19.1)</td>
</tr>
<tr>
<td>thread_safe</td>
<td>(0.3.5)</td>
</tr>
<tr>
<td>tilt</td>
<td>(2.0.1)</td>
</tr>
<tr>
<td>turbolinks</td>
<td>(2.5.3)</td>
</tr>
<tr>
<td>coffee-rails</td>
<td></td>
</tr>
<tr>
<td>tzinfo</td>
<td>(1.2.2)</td>
</tr>
<tr>
<td>thread_safe</td>
<td>(~&gt; 0.1)</td>
</tr>
<tr>
<td>tzinfo-data</td>
<td>(1.2015.7)</td>
</tr>
<tr>
<td>tzinfo</td>
<td>(&gt;= 1.0.0)</td>
</tr>
<tr>
<td>uglifier</td>
<td>(2.7.2)</td>
</tr>
<tr>
<td>execjs</td>
<td>(&gt;= 0.3.0)</td>
</tr>
<tr>
<td>json</td>
<td>(&gt;= 1.8.0)</td>
</tr>
<tr>
<td>PLATFORMS</td>
<td></td>
</tr>
<tr>
<td>x64-mingw32</td>
<td></td>
</tr>
<tr>
<td>DEPENDENCIES</td>
<td></td>
</tr>
<tr>
<td>chartkick</td>
<td></td>
</tr>
<tr>
<td>coffee-rails</td>
<td>(~&gt; 4.1.0)</td>
</tr>
<tr>
<td>descriptive_statistics</td>
<td>(~&gt; 2.4.0)</td>
</tr>
<tr>
<td>highcharts-rails</td>
<td>(~&gt; 0.3.18)</td>
</tr>
<tr>
<td>mysql2</td>
<td>(~&gt; 0.3.18)</td>
</tr>
<tr>
<td>rails</td>
<td>(= 4.2.1)</td>
</tr>
<tr>
<td>roo</td>
<td>(2.3.0)</td>
</tr>
<tr>
<td>sass-rails</td>
<td>(~&gt; 5.0)</td>
</tr>
<tr>
<td>sdoc</td>
<td>(~&gt; 0.4.0)</td>
</tr>
<tr>
<td>spreadsheet</td>
<td></td>
</tr>
<tr>
<td>turbolinks</td>
<td></td>
</tr>
<tr>
<td>tzinfo-data</td>
<td></td>
</tr>
<tr>
<td>uglifier</td>
<td>(&gt;= 1.3.0)</td>
</tr>
<tr>
<td>163</td>
<td></td>
</tr>
<tr>
<td>BUNDLED WITH</td>
<td></td>
</tr>
<tr>
<td>1.10.4</td>
<td></td>
</tr>
<tr>
<td>166</td>
<td></td>
</tr>
</tbody>
</table>
DATABASE SCHEMA:

Page 1 of 1
# encoding: UTF-8
# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate
this schema definition.
#
# Note that this schema.rb definition is the
authoritative source for your
# database schema. If you need
to create the application
database on another
# system, you should be using
db:schema:load, not running
all the migrations 9 # from
scratch. The latter is a flawed
and unsustainable approach
(the more migrations 10 #
you'll amass, the slower it'll
run and the greater likelihood
for issues).
#
# It's strongly recommended
that you check this file into
your version control system.

ActiveRecord::Schema.define(version: 20160718175503) do
  create_table "answers", force: :cascade do |t|
    t.integer  "enroll_id", limit: 4
    t.string   "question_text", limit: 255
    t.boolean  "correct", limit: 1
    t.string   "answer", limit: 255
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.integer "knowledge_topic_id", limit: 4
  end
  add_index "answers", ["enroll_id"], name: "index_answers_on_enroll_id",
  using: :btree
  add_index "answers", ["knowledge_topic_id"], name: "index_answers_on_knowledge_topic_id",
  using: :btree
  create_table "class_averages", force: :cascade do |t|
    t.string   "program", limit: 255
    t.integer "course", limit: 4
    t.string   "semester", limit: 255
    t.integer "year_offered", limit: 4
    t.integer "number_students", limit: 4
    t.float    "quiz_average", limit: 24
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end
  create_table "debug_pages", force: :cascade do |t|
    t.datetime "created_at", null: false
  end
end
create_table "enrolls", force: :cascade do |t|
  t.integer  "student_id", limit: 4
  t.integer  "course_registration_number", limit: 4
  t.string   "academic_progress", limit: 255
  t.string   "grade", limit: 255
  t.decimal  "quiz_score", precision: 10
  t.integer  "year_offered", limit: 4
  t.string   "semester", limit: 255
  t.integer  "section", limit: 4
  t.string   "faculty_last_name", limit: 255
  t.string   "faculty_first_name", limit: 255
  t.string   "course", limit: 255
  t.string   "teaching_style", limit: 255
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end
61
62  add_index "enrolls", ["student_id"], name: "index_enrolls_on_student_id", using: :btree
63
create_table "knowledge_topics", force: :cascade do |t|
  t.string   "knowledge_area", limit: 255
  t.string   "knowledge_unit", limit: 255
  t.string   "knowledge_topic", limit: 255
  t.integer  "year_added", limit: 4
  t.integer  "correct_answers", limit: 4
  t.integer  "incorrect_answers", limit: 4
  t.integer  "temp_correct_answer", limit: 4
  t.integer  "temp_incorrect_answer", limit: 4
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end
76
create_table "kts_covered_by_answers", force: :cascade do |t|
  t.integer  "knowledge_topic_id", limit: 4
  t.integer  "answer_id", limit: 4
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end
83
add_index "kts_covered_by_answers", ["answer_id"], name: "index_kts_covered_by_answers_on_answer_id", using: :btree
add_index
"kts_covered_by_answers",
["knowledge_topic_id"], name: "index_kts_covered_by_answers_on_knowledge_topic_id",
using: :btree
86
create_table
"load_internal_reports", force: :cascade do |t|
t.datetime "created_at", null: false
end
91
create_table
"load_slo_covered_by_kts", force: :cascade do |t|
t.datetime "created_at", null: false
end
96
create_table
"load_standard_reports", force: :cascade do |t|
t.datetime "created_at", null: false
end
101
create_table
"load_student_learning_outcomes",
force: :cascade do |t|
t.datetime "created_at", null: false
end
106
create_table
"load_student_quiz_results",
force: :cascade do |t|
datetime "created_at", null: false
datetime "updated_at", null: false
datetime "created_at", null: false
datetime "updated_at", null: false
end
111
create_table
"peo_covered_by_slos", force: :cascade do |t|
t.integer "student_learning_outcome_id", limit: 4
t.integer "program_educational_objective_id", limit: 4
datetime "created_at", null: false
datetime "updated_at", null: false
end
118
add_index
"peo_covered_by_slos",
["program_educational_objective_id"], name: "index_peo_covered_by_slos_on_program_educational_objective_id",
using: :btree
add_index
"peo_covered_by_slos",
["student_learning_outcome_id"], name: "index_peo_covered_by_slos_on_student_learning_outcome_id",
using: :btree
121
create_table "prerequisites", force: :cascade do |t|
t.string "course", limit: 255
t.integer "enroll_id", limit: 4
t.integer "year_taken", limit: 4
t.string "faculty_last_name", limit: 255
t.string "semester", limit: 255
add_index "prerequisites", ["enroll_id"], name: "index_prerequisites_on_enroll_id", using: :btree
create_table "program_educational_objectives", force: :cascade do |t|
  t.string   "accredidation_body", limit: 255
  t.string   "title", limit: 255
  t.string   "description", limit: 255
  t.integer  "year_added", limit: 4
  t.integer  "correct_answers", limit: 4
  t.integer  "incorrect_answers", limit: 4
  t.integer  "temp_correct_answer", limit: 4
  t.integer  "temp_incorrect_answer", limit: 4
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end

add_index "program_of_studies", ["enroll_id"], name: "index_program_of_studies_on_enroll_id", using: :btree
create_table "slo_averages", force: :cascade do |t|
  t.string   "program", limit: 255
  t.string   "title", limit: 255
  t.string   "semester", limit: 255
  t.integer  "year_offered", limit: 4
  t.integer  "number_students", limit: 4
  t.float    "average", limit: 24
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end

create_table "slo_covered_by_kts", force: :cascade do |t|
  t.integer  "student_learning_outcome_id", limit: 4
  t.integer  "knowledge_topic_id", limit: 4
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end
add_index "slo_covered_by_kts", ["knowledge_topic_id"], name: "index_slo_covered_by_kts_on_knowledge_topic_id", using: :btree
add_index "slo_covered_by_kts", ["student_learning_outcome_id"], name: "index_slo_covered_by_kts_on_student_learning_outcome_id", using: :btree

create_table "student_learning_outcomes", force: :cascade do |t|
t.string  "accredidation_body", limit: 255
  t.string  "title", limit: 255
  t.string  "description", limit: 255
  t.integer  "year_added", limit: 4
  t.integer  "correct_answers", limit: 4
  t.integer  "incorrect_answers", limit: 4
  t.integer  "temp_correct_answer", limit: 4
  t.integer  "temp_incorrect_answer", limit: 4
  t.datetime  "created_at", null: false
  t.datetime  "updated_at", null: false
end

add_foreign_key "answers", "enrolls"
add_foreign_key "answers", "knowledge_topics"
add_foreign_key "enrolls", "students"
add_foreign_key "kts_covered_by_answers", "answers"
add_foreign_key "kts_covered_by_answers", "knowledge_topics"
add_foreign_key "peo_covered_by_slos", "program_educational_objectives"
add_foreign_key "peo_covered_by_slos", "program_of_studies"
add_foreign_key "prerequisites", "enrolls"
add_foreign_key "program_of_studies", "enrolls"
add_foreign_key "slo_covered_by_kts", "knowledge_topics"
add_foreign_key
"slo_covered_by_kts",
"student_learning_outcomes"
end
217
**FILE**
D:\railsApps\playgroundJan2016\app\views\answers\new.html.erb

1 <h1>New Answer</h1>

2 <%= render 'form' %>

3 <%= link_to 'Back', answers_path %>

---

**FILE**
D:\railsApps\playgroundJan2016\app\views\answers\edit.html.erb

1 <h1>Editing Answer</h1>

2 <%= render 'form' %>

3 <%= link_to 'Show', @answer %>

4 <%= link_to 'Back', answers_path %>

---

**FILE**
D:\railsApps\playgroundJan2016\app\views\answers\show.html.erb

1 <p id="notice"><%= notice %></p>

2 <p><strong>Enroll:</strong><br><%= @answer.Enroll %></p>

3 <p><strong>Question text:</strong><br><%= @answer.question_text %></p>

4 <p><strong>Correct:</strong><br><%= @answer.correct %></p>

---

**FILE**
D:\railsApps\playgroundJan2016\app\views\answers\_form.html.erb

13 <div class="field"><%= f.label :Enroll_id %><br><%= f.text_field :Enroll_id %></div>

---

14 <div class="field"><%= f.label :question_text %><br><%= f.text_field :question_text %></div>

---

15 <div class="field"><%= f.label :correct %><br><%= f.text_field :correct %></div>
<%= f.check_box :correct %>
</div>
<div class="field">
<%= f.label :answer %><br>
<%= f.text_field :answer %>
</div>
<div class="actions">
<%= f.submit %>
</div>
</div>
</div>

File
D:\railsApps\playgroundJan2016\app\views\answers\index.html.erb

1
<p id="notice"><%= notice %></p>
2
3 <h1>Listing Answers</h1>
4
<table>
<thead>
<tr>
<th>Course</th>
<th>Student</th>
<th>Question text</th>
<th>Correct</th>
<th>Answer</th>
</tr>
</thead>
16
<tbody>
<% @answers.each do |answer| %>
<tr>
<td>
<% if answer.enroll_id %>
<%= Enroll.find_by(id: answer.enroll_id).course %>
<% else %>
<%= answer.Enroll %>
<% end %>
</td>
<td>
<% if answer.enroll_id %>
<%= Student.find_by(id: Enroll.find_by(id: answer.enroll_id).student_id).w_number %>
<% else %>
<%= answer.Enroll %>
<% end %>
</td>
<td><%= answer.question_text %></td>
<td><%= answer.correct %></td>
<td><%= answer.answer %></td>
<td><%= answer.knowledge_topic_id %></td>
</tr>
<% end %>
</tbody>
</table>

45
46
47
<%= link_to 'Show', answer %>
</td>
41 </tr>
</tbody>
</table>
45
46 <br>
47
48 <%= link_to 'New Answer', new_answer_path %>
49

File
D:\railsApps\playgroundJan2016\app\views\answers\show.json.jbuilder
1 json.extract! @answer, :id, :Enroll_id, :question_text, :correct, :answer, :created_at, :updated_at
2
File
D:\railsApps\playgroundJan2016\app\views\enrolls\new.html.erb
1 <h1>New Enroll</h1>
2
3 <%= render 'form' %>
4
5 <%= link_to 'Back', enrolls_path %>
6

File
D:\railsApps\playgroundJan2016\app\views\enrolls\edit.html.erb
1 <h1>Editing Enroll</h1>
2
3 <%= render 'form' %>
4
5 <%= link_to 'Show', @enroll %> |
6 <%= link_to 'Back', enrolls_path %>
7

File
D:\railsApps\playgroundJan2016\app\views\enrolls\show.html.erb
1 <p id="notice"><%= notice %></p>
2
3 <strong>Student:</strong> <%= @enroll.Student %>
7
3 <strong>Course registration number:</strong> <%= @enroll.course_registration_number %>
12

<strong>Academic progress:</strong>
<%= @enroll.academic_progress %>
17
<%= @enroll.grade %>
22
<%= @enroll.quiz_score %>
27
<%= @enroll.year_offered %>
32
<%= @enroll.semester %>
37
<%= @enroll.section %>
42
33 <strong>Faculty last name:</strong> <%= @enroll.faculty_last_name %>
<%= @enroll.facebook_last_name %>
<%= @enroll.facebook_first_name %>
Course:  

Teaching style:  

Edit:  
Back:
app\views\enrolls\_form.html.erb

```html
<% if faculty_first_name %>
<%= f.label :faculty_first_name %><br>
<%= f.text_field :faculty_first_name %>
</div>
<% end %>

<% if course %>
<%= f.label :course %><br>
<%= f.text_field :course %>
</div>
<% end %>

<% if teaching_style %>
<%= f.label :teaching_style %><br>
<%= f.text_field :teaching_style %>
</div>
<% end %>

<%= f.submit %>
</div>
```

File:
D:\railsApps\playgroundJan2016\app\views\enrolls\index.html.erb

```html
1 <p id="notice">%= notice %></p>
2
3 <h1>Listing Enrolls</h1>
4 <table>
  <thead>
    <tr>
      <th>Student</th>
      <th>Course registration number</th>
      <th>Academic progress</th>
      <th>Grade</th>
      <th>Quiz score</th>
      <th>Year offered</th>
      <th>Semester</th>
      <th>Section</th>
      <th>Faculty last name</th>
      <th>Faculty first name</th>
      <th>Course</th>
      <th>Teaching style</th>
      <th colspan="3"></th>
    </tr>
  </thead>
  <tbody>
  <% @enrolls.each do |enroll| %>
    <tr>
      <td><%= if enroll.student_id %>
        <%= Student.find_by(id: enroll.student_id).w_number %>
      <% else %>
        <%= enroll.student_id %>
      <% end %></td>
      <td><%= enroll.course_registration_number %></td>
      <td><%= enroll.academic_progress %></td>
      <td><%= enroll.grade %></td>
      <td><%= enroll.quiz_score %></td>
      <td><%= enroll.year_offered %></td>
      <td><%= enroll.semester %></td>
      <td><%= enroll.section %></td>
      <td><%= enroll.faculty_last_name %></td>
      <td><%= enroll.faculty_first_name %></td>
      <td><%= enroll.course %></td>
      <td><%= enroll.teaching_style %></td>
      <td><%= link_to 'Show', enroll %></td>
      <td><%= link_to 'Edit', edit_enroll_path(enroll) %></td>
    </tr>
  <% end %>
  ```
<td><%= link_to 'Destroy', enroll, method: :delete, data: { confirm: 'Are you sure?' } %></td>
</tr>
</tbody>

File - D:\railsApps\playgroundJan2016\app\views\enrolls\index.html.erb

<%= link_to 'New Enroll', new_enroll_path %>
File
D:\railsApps\playgroundJan2016\app\views\layouts\application.html.erb
<!DOCTYPE html>
<html>
<head>
<title>PlaygroundJan2016</title>
<%= csrf_meta_tags %>
</head>
<body>
8
9 <%= yield %>
10
</body>
</html>
Wright State's Computer Science and Engineering Department

Continuous Assessment System

Loading Data

Continuous Assessment System

Standard Reports

Data Sheets
Reseting the Database

DELETES STUDENT QUIZ RESULTS

```
<%=
  link_to 'Cleanup', '/cleanup'
<% end
```

File - D:\railsApps\playgroundJan2016\app\views\students\new.html.erb

```erb
1 <h1>New Student</h1>
2 <%= render 'form' %>
3
4 5 <%= link_to 'Back', students_path %>
6```
VIEWS: STUDENTS

File
D:\railsApps\playgroundJan2016\app\views\students\edit.html.erb
1 <h1>Editing Student</h1>
2 3 <%= render 'form' %
4 <%= link_to 'Show', @student %>
|\n5 <%= link_to 'Back', students_path %> 7

File
D:\railsApps\playgroundJan2016\app\views\students\show.html.erb
1 <p id="notice"><%= notice %></p>
2
3 <strong>W number:</strong>
4 <%= @student.w_number %>
</p>
7
8 <strong>Last name:</strong>
9 <%= @student.last_name %>
</p>
12
<p>
9 <strong>First name:</strong>
10 <%= @student.first_name %>
</p>
17
<p>
9 <strong>Gender:</strong>
10 <%= @student.gender %>
</p>
22
<p>
9 <strong> Ethnicity:</strong>
10 <%= @student.ethnicity %>
</p>
27

File
D:\railsApps\playgroundJan2016\app\views\students\_form.html.erb
27
28 <strong>High school gpa:</strong>
29 <%= @student.high_school_gpa %>
30
31
32
33 <strong>Act:</strong>
34 <%= @student.act %>
</p>
37
<p>
38 <strong>Mpl:</strong>
39 <%= @student.mpl %>
</p>
42
<p>
43 <strong>Act math:</strong>
44 <%= @student.act_math %>
</p>
47
<p>
48 <strong>Path 1160:</strong>
Page 1 of 2

File
D:\railsApps\playgroundJan2016\app\views\students\show.html.erb
1
2
3 <%= link_to 'Edit', edit_student_path(@student) %>
|\n4 <%= link_to 'Back', students_path %> 7

File
D:\railsApps\playgroundJan2016\app\views\students\_form.html.erb
1
2
3 <% if @student.errors.any? %>
4 <div id="error_explanation">
5 <h2>Pluralize(@student.errors.count
6 %}
prohibited this student from being saved:

<ul>
  <% @student.errors.full_messages.each do |message| %>
    <li><%= message %></li>
  <% end %>
</ul>
</div>
</div>

<% if @student.errors.empty? %>
</div>
</div>

<% end %>

13

<div class="field">
  <%= f.label :w_number %><br>
  <%= f.text_field :w_number %>
</div>

<div class="field">
  <%= f.label :last_name %><br>
  <%= f.text_field :last_name %>
</div>

<div class="field">
  <%= f.label :first_name %><br>
  <%= f.text_field :first_name %>
</div>

<div class="field">
  <%= f.label :gender %><br>
  <%= f.text_field :gender %>
</div>

<div class="field">
  <%= f.label :ethnicity %><br>
  <%= f.text_field :ethnicity %>
</div>

<div class="field">
  <%= f.label :high_school_gpa %><br>
  <%= f.text_field :high_school_gpa %>
</div>

<div class="field">
  <%= f.label :act %><br>
  <%= f.number_field :act %>
</div>

<div class="field">
  <%= f.label :mpl %><br>
  <%= f.number_field :mpl %>
</div>

<div class="field">
  <%= f.label :act_math %><br>
  <%= f.number_field :act_math %>
</div>

<% if @student.errors.empty? %>
</div>
</div>

<% end %>

57

58

Page 2 of 2
Listing Students

<table>
<thead>
<tr>
<th>W number</th>
<th>Last name</th>
<th>First name</th>
<th>Gender</th>
<th>Ethnicity</th>
<th>High school gpa</th>
<th>Act</th>
<th>MPL</th>
<th>Act math</th>
<th>Path 1160</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

{% @students.each do |student| %}
  <tr>
    <td><%= student.w_number %></td>
    <td><%= student.last_name %></td>
    <td><%= student.first_name %></td>
    <td><%= student.gender %></td>
    <td><%= student.ethnicity %></td>
    <td><%= student.high_school_gpa %></td>
    <td><%= student.act %></td>
    <td><%= student.mpl %></td>
    <td><%= student.act_math %></td>
    <td><%= student.path_1160 %></td>
  </tr>
{% end %}

{% link_to 'Show', student %>
{% link_to 'Edit', edit_student_path(student) %>
{% link_to 'Destroy', student, method: :delete, data: 
  { confirm: 'Are you sure?' } %>

{% link_to 'New Student', new_student_path %>

1 <h1>New Slo Average</h1>
2
3 <%= render 'form' %>
4
5 <%= link_to 'Back', slo_averages_path %>

File
D:\railsApps\playgroundJan2016\app\views\slo_averages\edit.html.erb
1 <h1>Editing Slo Average</h1>
2
3 <%= render 'form' %>
4
5 <%= link_to 'Show', @slo_average %>
6
7 <%= link_to 'Back', slo_averages_path %>

File
D:\railsApps\playgroundJan2016\app\views\slo_averages\show.html.erb
1 <p id="notice"><%= notice %></p>
2
3 <p><strong>Program:</strong><%= @slo_average.program %></p>
4
5 <p><strong>Title:</strong><%= @slo_average.title %></p>
6
7
8 <p><strong>Semester:</strong><%= @slo_average.semester %></p>
9
10 <p><strong>Year offered:</strong><%= @slo_average.year_offered %></p>
11
12 <p><strong>Number students:</strong><%= @slo_average.number_students %></p>
13
14 <p><strong>Average:</strong><%= @slo_average.average %></p>
15
16 <p><%= link_to 'Edit', edit_slo_average_path(@slo_average) %></p>
17
18 <%= link_to 'Back', slo_averages_path %>
19
20
21 <strong>Prohibited this slo_average from being saved:</strong>
22
23 <li><%= message %></li>
<th>Program</th>
<th>Title</th>
<th>Semester</th>
<th>Year offered</th>
<th>Number students</th>
<th>Average</th>
<th colspan="3"></th>
</tr>
</thead>
<tbody>
  <tr><td><%= slo_average.program %></td>
   <td><%= slo_average.title %></td>
   <td><%= slo_average.semester %></td>
   <td><%= slo_average.year_offered %></td>
   <td><%= slo_average.number_students %></td>
   <td><%= slo_average.average %></td>
   <td><%= link_to 'Show', slo_average %></td>
   <td><%= link_to 'Edit', edit_slo_average_path(slo_average) %></td>
   <td><%= link_to 'Destroy', slo_average, method: :delete, data: { confirm: 'Are you sure?' } %></td>
  </tr>
  <% end %>
</tbody>
</table>

<br>
<a href="#">New Slo average</a>
</body>
</html>
```ruby
File
D:\railsApps\playgroundJan2016\app\views\slo_averages\show.jbuilder
1  json.extract! @slo_average,
   :id, :program, :title,
   :semester, :year_offered,
   :number_students, :average,
   :created_at, :
updated_at
2

File
D:\railsApps\playgroundJan2016\app\views\slo_averages\index.jbuilder
json.array!(@slo_averages) do |slo_average|
 json.extract! slo_average, :id,
   :program, :title, :semester,
   :year_offered,
   :number_students, :average
 json.url
 slo_average_url(slo_average,
 format: :json)
end
```
# VIEWS: PREREQUISITES

File  
D:\railsApps\playgroundJan2016\app\views\prerequisites\new.html.erb  
1 <h1>New Prerequisite</h1>  
2 <%= render 'form' %>  
3 <%= link_to 'Back', prerequisites_path %>  

File  
D:\railsApps\playgroundJan2016\app\views\prerequisites\edit.html.erb  
1 <h1>Editing Prerequisite</h1>  
2 <%= render 'form' %>  
3 <%= link_to 'Show', @prerequisite %> | <%= link_to 'Back', prerequisites_path %>  

File  
D:\railsApps\playgroundJan2016\app\views\prerequisites\show.html.erb  
1 <p id="notice"><%= notice %></p>  
2 <p>  
<%= @prerequisite.year_taken %>  
</p>  
7  
<p>  
<%= @prerequisite.faculty_last_name %>  
</p>  
17  
<p>  
<%= @prerequisite.semester %>  
</p>  
22  
<p>  
<%= link_to 'Edit', edit_prerequisite_path(@prerequisite) %> |  
<%= link_to 'Back', prerequisites_path %>  
30  
File  
D:\railsApps\playgroundJan2016\app\views\prerequisites\_form.html.erb  
13  
<div class="field">

Digital Assistant
<%= f.label :course %><br>
<%= f.text_field :course %>
</div>
<div class="field">
<%= f.label :Enroll_id %><br>
<%= f.text_field :Enroll_id %>
</div>
<div class="field">
<%= f.label :year_taken %><br>
<%= f.number_field :year_taken %>
</div>
<div class="field">
<%= f.label :faculty_last_name %><br>
<%= f.text_field :faculty_last_name %>
</div>
<div class="field">
<%= f.label :semester %><br>
<%= f.text_field :semester %>
</div>
<div class="actions">
<%= f.submit %>
</div>
</div>

<%= link_to 'Show', prerequisite %>
<%= link_to 'Edit', edit_prerequisite_path(prerequisite) %>
<%= link_to 'Destroy', prerequisite, method: :delete, data: { confirm: 'Are you sure?' } %>

<% end %>
</tbody>
</table>

<%= link_to 'New Prerequisite', new_prerequisite_path %>

Prerequisites</h1>

Course
Enroll
Year taken
Faculty last name
Semester

colspan="3"

Listing Prerequisites</h1>

<tr><th>Course</th><th>Enroll</th><th>Year taken</th><th>Faculty last name</th><th>Semester</th>
<td colspan="3"></td>
</tr>

1 <p id="notice"><%= notice %></p>
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
VIEWS: CLASS AVERAGES

File  
D:\railsApps\playgroundJan2016\app\views\class_averages\new.html.erb  
1 <h1>**New Class Average**</h1>  
2  
3 <%= render 'form' %>  
4  
5 <%= link_to 'Back', class_averages_path %>  

File  
D:\railsApps\playgroundJan2016\app\views\class_averages\edit.html.erb  
1 <h1>**Editing Class Average**</h1>  
2  
3 <%= render 'form' %>  
4  
5 <%= link_to 'Show', @class_average %> |  
6 <%= link_to 'Edit', edit_class_average_path(@class_average) %> |  
7 <%= link_to 'Back', class_averages_path %>  

File  
D:\railsApps\playgroundJan2016\app\views\class_averages\show.html.erb  
1 <p id="notice">% notice %</p>  
2  
3 <strong>**Program:**</strong>  
4 <%= @class_average.program %>  
5 </p>  
6  
7 <strong>**Course:**</strong>  
8 <%= @class_average.course %>  
9 </p>  
10  
11 <strong>**Semester:**</strong>  
12 <%= @class_average.semester %>  
13 </p>  
14  
15 <strong>**Year offered:**</strong>  
16 <%= @class_average.year_offered %>  
17 </p>  
18  
19 <strong>**Number students:**</strong>  
20 <%= @class_average.number_students %>  
21 </p>  
22  
23 <strong>**Quiz average:**</strong>  
24 <%= @class_average.quiz_average %>  
25 </p>  
26  
27 <%= link_to 'Edit', edit_class_average_path(@class_average) %> |  
28 <%= link_to 'Back', class_averages_path %>  
29  
30 File  
D:\railsApps\playgroundJan2016\app\views\class_averages\_form.html.erb  
1 <%= form_for(@class_average) do |f| %&gt;  
2 |  
3 <%= if @class_average.errors.any? %&gt;  
4 &lt;%= @class_average.errors.count, "error" %&gt; prohibited this class_average from being saved:  
5 &lt;/h2&gt;  
6 &lt;ul&gt;  
7 &lt;/ul&gt;
<@class_average.errors.full_messages.each do |message| %>
	<li><%= message %></li>
<% end %>
</ul>
<% end %>
13
<div class="field">
  <%= f.label :program %><br>
  <%= f.text_field :program %>
</div>
<div class="field">
  <%= f.label :course %><br>
  <%= f.number_field :course %>
</div>
<div class="field">
  <%= f.label :semester %><br>
  <%= f.text_field :semester %>
</div>
<div class="field">
  <%= f.label :year_offered %><br>
  <%= f.number_field :year_offered %>
</div>
<div class="field">
  <%= f.label :number_students %><br>
  <%= f.number_field :number_students %>
</div>
<div class="field">
  <%= f.label :quiz_average %><br>
  <%= f.text_field :quiz_average %>
</div>
<div class="actions">
  <%= f.submit %>
</div>
<% end %>

File D:\railsApps\playgroundJan2016\app\views\class_averages\index.html.erb
1 <p id="notice"><%= notice %></p>
2

3 <h1>Listing Class Averages</h1>
4 <table>
  <thead>
    <tr>
      <th>Program</th>
      <th>Course</th>
      <th>Semester</th>
      <th>Year offered</th>
      <th>Number students</th>
      <th>Quiz average</th>
    </tr>
  </thead>
  <tbody>
    <%= @class_averages.each do |class_average| %>
    <tr>
      <td><%= class_average.program %></td>
      <td><%= class_average.course %></td>
      <td><%= class_average.semester %></td>
      <td><%= class_average.year_offered %></td>
      <td><%= class_average.number_students %></td>
      <td><%= class_average.quiz_average %></td>
      <td><%= link_to 'Show', class_average %></td>
      <td><%= link_to 'Edit', edit_class_average_path(class_average) %></td>
      <td><%= link_to 'Destroy', class_average, method: :delete, data: { confirm: 'Are you sure?' } %></td>
    </tr>
    <% end %>
  </tbody>
</table>
34 35 <br>
36 37 <%= link_to 'New Class average', 
new_class_average_path %> 38 
File
D:\railsApps\playgroundJan2016\ 
app\views\class_averages\show.j 
son.jbuilder

1 json.extract! @class_average, 
:id,     :program,     :course, 
:semester,     :year_offered, 
:number_students, 
:quiz_average, 
:created_at, :updated_at
2

File
D:\railsApps\playgroundJan2016\ 
app\views\class_averages\index. 
json.jbuilder

1 json.array!(@class_averages) do 
|class_average|
2 json.extract! class_average, 
:id,     :program,     :course, 
:semester,     :year_offered, 
:number_students, :quiz_average 
json.url 
class_average_url(class_average, format: :json)
end
5
File
D:\railsApps\playgroundJan2016\app\views\knowledge_topics\new.html.erb
1 <h1>New Knowledge Topic</h1>
2 <%= render 'form' %>
3 <%= link_to 'Back', knowledge_topics_path %>

File
D:\railsApps\playgroundJan2016\app\views\knowledge_topics\edit.html.erb
1 <h1>Editing Knowledge Topic</h1>
2 <%= render 'form' %>
3 <%= link_to 'Show', @knowledge_topic %> | <%= link_to 'Back', knowledge_topics_path %>

File
D:\railsApps\playgroundJan2016\app\views\knowledge_topics\show.html.erb
1 <p id="notice"><%= notice %></p>
2 <p><strong>Knowledge area:</strong><%= @knowledge_topic.knowledge_area %></p>
3 <p><strong>Knowledge unit:</strong><%= @knowledge_topic.knowledge_topic %></p>
4 <p><strong>Year added:</strong><%= @knowledge_topic.year_added %></p>
5 <p><strong>Active:</strong><%= @knowledge_topic.active %></p>
6 <p><strong>Correct answers:</strong><%= @knowledge_topic.correct_answers %></p>
7 <p><strong>Incorrect answers:</strong><%= @knowledge_topic.incorrect_answers %></p>
8 <p><strong>Temp correct answer:</strong><%= @knowledge_topic.temp_correct_answer %></p>
Temp incorrect answer:

```
<%= @knowledge_topic.temp_incorrect_answer %>
```

47

```
<%= link_to 'Edit', edit_knowledge_topic_path(@knowledge_topic) %>
<%= link_to 'Back', knowledge_topics_path %>
```

File

D:\railsApps\playgroundJan2016\app\views\knowledge_topics\_form.html.erb

```
<%= form_for(@knowledge_topic) do |f| %>
  <% if @knowledge_topic.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@knowledge_topic.errors.count, "error") %>
      prohibited this knowledge_topic from being saved:</h2>
      <ul>
        <% @knowledge_topic.errors.full_messages.each do |message| %>
        <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
  <br>
  <%= f.label :knowledge_area %><br>
  <%= f.text_field :knowledge_area %>
  <br>
  <%= f.label :knowledge_topic %><br>
  <%= f.text_field :knowledge_topic %>
  <br>
  <%= f.label :year_added %><br>
  <%= f.number_field :year_added %>
  <br>
  <%= f.label :active %><br>
  <%= f.check_box :active %>
  <br>
  <%= f.label :correct_answers %><br>
  <%= f.number_field :correct_answers %>
  <br>
  <%= f.label :incorrect_answers %><br>
  <%= f.number_field :incorrect_answers %>
  <br>
  <%= f.label :temp_correct_answer %><br>
  <%= f.number_field :temp_correct_answer %>
  <br>
  <%= f.label :temp_incorrect_answer %><br>
  <%= f.number_field :temp_incorrect_answer %>
</div>
```

Page 1 of 2
File
D:\railsApps\playgroundJan2016\app\views\knowledge_topics\_form.html.erb
</div>
<div class="actions">
  <%= f.submit %>
</div>

File
D:\railsApps\playgroundJan2016\app\views\knowledge_topics\index.html.erb
1 <p id="notice"><%= notice %></p>
2
3  <h1>Listing Knowledge Topics</h1>
4
5  <table>
6    <thead>
7      <tr>
8        <th>ID</th>
9        <th>Knowledge area</th>
10       <th>Knowledge unit</th>
11       <th>Knowledge topic</th>
12       <th>Year added</th>
13       <th>Correct answers</th>
14       <th>Incorrect answers</th>
15       <th>Temp correct answer</th>
16       <th>Temp incorrect answer</th>
17       <th colspan="3"></th>
18      </tr>
19    </thead>
20    <tbody>
21      <% @knowledge_topics.each do |knowledge_topic| %>
22         <tr>
23           <td><%= knowledge_topic.id %></td>
24           <td><%= knowledge_topic.knowledge_area %></td>
25           <td><%= knowledge_topic.knowledge_unit %></td>
26           <td><%= knowledge_topic.knowledge_topic %></td>
27           <td><%= knowledge_topic.year_added %></td>
28           <td><%= knowledge_topic.correct_answers %></td>
29           <td><%= knowledge_topic.incorrect_answers %></td>
30           <td><%= knowledge_topic.temp_correct_answer %></td>
31           <td><%= knowledge_topic.temp_incorrect_answer %></td>
32           <td><%= link_to 'Show', knowledge_topic %></td>
33           <td><%= link_to 'Edit', edit_knowledge_topic_path(knowledge_topic) %></td>
34           <td><%= link_to 'Destroy', knowledge_topic, method: :delete, data: { confirm: 'Are you sure?' } %></td>
35         </tr>
36      <% end %>
37    </tbody>
38  </table>
39 40 41 <br>
42 43 <%= link_to 'New Knowledge topic', new_knowledge_topic_path %>
44
File
D:\railsApps\playgroundJan2016\app\views\knowledge_topics\show.json.jbuilder
1 json.extract! @knowledge_topic, :id, :knowledge_area,
json.array!(@knowledge_topics)
do |knowledge_topic|
  json.extract! knowledge_topic,
  :id, :knowledge_area,
  :knowledge_unit, :knowledge_topic, :year_added, :active,
  :correct_answers, :incorrect_answers,
  :temp_correct_answer, :temp_incorrect_answer
  json.url knowledge_topic_url(knowledge_topic, format: :json)
end
File  
D:\railsApps\playgroundJan2016\app\views\standard_reports\index.html.erb
<h1><b>Search Database Test</b></h1>

<h2><b>Search: </b></h2>
<form action="/standard_reports/search" method="get">
  <label>Query with first name:</label>
  <input type="text" name="qry">
  <button type="submit">Query</button>
</form>

<form action="/standard_reports/abet_report" method="get">
  <button type="submit">ABET Report</button>
</form>

File  
D:\railsApps\playgroundJan2016\app\views\standard_reports\table.html.erb
<table style="width:75%",
  align="center">
  <thead>
    <tr>
      % headers_list.each do |title| %
      <th><%= title %></th>
      % end %
    </tr>
  </thead>
  <tbody>
    % table_data.each do |row| %
    <tr>
      % row.each do |column| %
      <td><%= column %></td>
      % end %
    </tr>
    % end %
  </tbody>
</table>

File  
D:\railsApps\playgroundJan2016\app\views\standard_reports\answers.html.erb
<h1><b>Listing answers</b></h1>

<script type="text/javascript" charset="UTF-8">
$(function () {
  $('#container').highcharts({
    chart: {
      type: 'bar'
    },
    title: {
      text: 'Answered Correctly'
    },
    xAxis: {
      categories: ['CS 3100 Answers']
    },
    yAxis: {
      title: {
        text: 'Fruit eaten'
      }
    },
    series: [{
      headers_list.each do |title| %
      <th><%= title %></th>
      % end %
    </tr>
  </thead>
  <tbody>
    % table_data.each do |row| %
    <tr>
      % row.each do |column| %
      <td><%= column %></td>
      % end %
    </tr>
    % end %
  </tbody>
</table>
name: 'Right',
data: [<%= @correct.number %>] },
name: 'Wrong',
data: [<%= @incorrect.number %>] ]
});
});</script>
32
33 <div id="container" style="width:100%;
height:400px;"></div>
34
35 <table>
36 <thead>
37 <tr>
38 <th>Answer</th>
39 <th>Is correct</th>
40 <th>Number Students Select</th>
41 <th colspan="3"></th>
42 </tr>
43 </thead>
44 <tbody>
45 @answers.each do |answer|
46  <tr>
47  <td><%= answer.answer %></td>
48  <td><%= answer.is_correct %></td>
49  <td><%= answer.number_students_selected %></td>
50  </tr>
51 end
52 </tbody>
53 </table>
54 <br>
55 <script type="text/javascript" charset="UTF-8">
$(function () {

$("#<%= function_name %>").highcharts({
  title: {
    text: '<%=title%'
  },
  legend: {
    enabled: true
  },
  xAxis: {
    categories: <%= raw table[0] %>,
    title: {
      text: 'Course'
    }
  },
  yAxis: {
    title: {
      text: 'Average Grades for Courses'
    }
  },
  series: [
    {
      name: 'Quartiles',
      type: 'boxplot',
      data: [
        <%= raw table[i] %>,
        <%= raw table[table.length - 1] %>
      ]
    },
    {
      name: 'Past Terms',
      type: 'scatter',
      color: 'rgba(0, 0, 0, 1)',
      data: <%= raw past_data_table %>
    }
  ]
});

$("#%<%= poll_id %>").highcharts({
  title: {
    text: '<%=poll_title%'
  },
  legend: {
    enabled: true
  },
  xAxis: {
    categories: '<%= options %>',
    title: {
      text: 'Course'
    }
  },
  yAxis: {
    title: {
      text: 'Average Grades for Courses'
    }
  },
  series: [
    {
      name: 'Quartiles',
      type: 'boxplot',
      data: [
        <%= raw x %>,
        <%= raw y %>,
        <%= raw z %>,
        <%= raw w %>
      ]
    },
    {
      name: 'Past Terms',
      type: 'scatter',
      color: 'rgba(0, 0, 0, 1)',
      data: <%= raw past_data_table %>
    }
  ]
});

$("#%<%= poll_id %>").highcharts({
  title: {
    text: '<%=poll_title%'
  },
  legend: {
    enabled: true
  },
  xAxis: {
    categories: '<%= options %>',
    title: {
      text: 'Course'
    }
  },
  yAxis: {
    title: {
      text: 'Average Grades for Courses'
    }
  },
  series: [
    {
      name: 'Quartiles',
      type: 'boxplot',
      data: [
        <%= raw x %>,
        <%= raw y %>,
        <%= raw z %>,
        <%= raw w %>
      ]
    },
    {
      name: 'Past Terms',
      type: 'scatter',
      color: 'rgba(0, 0, 0, 1)',
      data: <%= raw past_data_table %>
    }
  ]
});
</script>
210
The richer data set of the new assessment infrastructure is maintained in a database allowing for specific drill down comparisons to compare different course preparation pathways, pedagogical styles, or any other variable of potential impact. For example, we can specifically address the question, is there a different in SLO achievement for students that take the two-semester intro-dictory computer science sequence for fully prepared incoming freshman (CS1180, CS1181) versus students that take the three course sequence for less prepared incoming students (CS1160, CS1161, CS1181).
student_learning_outcomes_year_percentages}

<%= render partial: 'bar_chart_grouped_numbers', locals: {title: 'sloStackYear', chart_title: 'SLOs by Years (number of questions answered)', table: @slos_stackable_years} %>

<!-- <%= render partial: "table", locals: {headers_list: ['a', 'b', 'c', 'd', 'e', 'f'], table_data: @slos_stackable_years} %> -->

<!-- <%= render partial: "bar_chart", locals: {title: 'sloYear', chart_title: 'SLOs by Years (number of questions answered)', table: @student_learning_outcomes_year, start_headers: 1, end_headers: 2, incorrect_data: 3, correct_data: 4} %> -->

<!-- <%= render partial: "table", locals: {headers_list: ['Semester', 'Year', 'SLO', 'Incorrect', 'Correct', 'Total Answers'], table_data: @student_learning_outcomes_year} %> -->

28

<h2>KNOWLEDGE AREA</h2>

<%= render partial: 'line_chart', locals: {title: 'kaLineYear', chart_title: 'Knowledge Areas by Years (% correct)', table: @knowledge_areas_year_percentages} %>

<%= render partial: 'bar_chart_grouped_numbers', locals: {title: 'kaStackYear', chart_title: 'Knowledge Areas by Years (number of questions answered)', table: @ka_stackable_years} %>

<!-- <%= render partial: "bar_chart", locals: {title: 'kaYearTest', chart_title: 'Knowledge Areas by Years (number of questions answered)', table: @knowledge_areas_year, start_headers: 1, end_headers: 2, incorrect_data: 3, correct_data: 4} %> -->

<!-- <%= render partial: "table", locals: {headers_list: ['Semester', 'Year', 'KA', 'Incorrect', 'Correct', 'Total Answers'], table_data: @knowledge_areas_year} %> -->

34

<h2>KNOWLEDGE UNIT</h2>

<%= render partial: "bar_chart", locals: {title: 'kuYear', chart_title: 'Knowledge Units by Years', table: @knowledge_units_year, start_headers: 1, end_headers: 3, incorrect_data: 4, correct_data: 5} %>

<!-- <%= render partial: "table", locals: {headers_list: ['Semester', 'Year', 'KA', 'KU', 'Incorrect', 'Correct', 'Total Answers'], table_data: @knowledge_units_year} %> -->

38

<h2>KNOWLEDGE TOPIC</h2>

<%= render partial: "bar_chart", locals: {title: 'ktYear', chart_title: 'Knowledge Topics by Years', table: @knowledge_topics_year, start_headers: 1, end_headers: 3, incorrect_data: 4, correct_data: 5} %>
<!----%=
render partial: "table", locals: {
headers_list: [
'Semester', 'Year', 'KA', 'KT',
'Incorrect', 'Correct', 'Total Answers'],
		table_data: @knowledge_topics_year} %>  ->
42
<!----<h2>Different Pathways</h2>  ->
<!----%=
render partial: "table", locals: {
headers_list: ['Semester', 'Year', 'KA', 'KT',
'1160 Incorrect', '1160 Correct', '1160 Total Answers',
'1180 Incorrect', '1180 Correct', '1180 Total Answers'],
		table_data: @different_programming_path_year} %>  ->
46
47 <h1>Data Separated By Semester</h1>
48
<h2>Student Learning Outcomes</h2>
%=
render partial: "line_chart", locals: {
title: 'sloLineTerm', chart_title: 'SLOs by Term',
table: @student_learning_outcomes_all_percentages} %>
%=
render partial: "bar_chart", locals: {
title: 'sloTerm', chart_title: 'SLOs by Term',
table: @student_learning_outcomes_all, start_headers: 0, end_headers: 2, correct_data: 4} %>  ->
<!----%=
render partial: "table", locals: {
headers_list: ['Semester', 'Year', 'SLO', 'Incorrect', 'Correct', 'Total Answers'],
		table_data: @student_learning_outcomes_all} %>  ->
53
54
<h2>KNOWLEDGE AREA</h2>
<!----%=
render partial: "bar_chart", locals: {
title: 'kaTerm', chart_title: 'Knowledge Areas by Term',
table: @knowledge_areas_all, start_headers: 0, end_headers: 2, incorrect_data: 3, correct_data: 4} %>  ->
<!----%=
render partial: "table", locals: {
headers_list: ['Semester', 'Year', 'KA', 'KU',
'Incorrect', 'Correct', 'Total Answers'],
		table_data: @knowledge_areas_all} %>  ->
58
<h2>KNOWLEDGE UNIT</h2>
<!----%=
render partial: "bar_chart", locals: {
title: 'kuTerm', chart_title: 'Knowledge Units by Term',
table: @knowledge_units_all, start_headers: 0, end_headers: 3, incorrect_data: 4, correct_data: 5} %>  ->
<!----%=
render partial: "table", locals: {
headers_list: ['Semester', 'Year', 'KA', 'KU',
'Incorrect', 'Correct', 'Total Answers'],
		table_data: @knowledge_units_all} %>  ->
62
<h2>KNOWLEDGE TOPIC</h2>
<!----%=
render partial: "bar_chart", locals: {
title: 'ktTerm', chart_title: 'Knowledge Topics by Term',
table: @knowledge_topics_all, start_headers: 0, end_headers: 3, incorrect_data: 4, correct_data: 5} %>  ->
<!----%=
render partial: "table", locals: {
headers_list: ['Semester', 'Year', 'KA', 'KT',
'Incorrect', 'Correct', 'Total Answers'],
		table_data: @knowledge_topics_all} %>  ->
67
This notes the course in which the knowledge topic is tested. It would have been learned in a previous course.

<h2>Computer Science Program</h2>

@@@slo_coverage_cs.each do |slo| @@
  @slo_coverage_ceg.each do |item|
    current_course = item[0]
    current_slo = StudentLearningOutcome.find(row.student_learning_outcome_id)
    if (slo.title.include?(current_slo.title) && (current_slo.accredidation_body.include?('CAC'))) =>
      kt =KnowledgeTopic.find(item[1])
      kt.knowledge_area = kt.knowledge_unit = kt.knowledge_topic
    end
  end
end

<h2>Computer Engineering Program</h2>

50 <%= link_to 'Home Page', root_path %>
SUMMARY REPORT

A summary of how students performed in prerequisite tests for the current term.

COURSE SUMMARY

A Value of 105 is assigned to courses that data wasn’t collected on this term.

<table>
<thead>
<tr>
<th>Course</th>
<th>Current Number Students</th>
<th>Current Quiz average</th>
<th>Number Past Terms</th>
<th>Past average</th>
<th>Std Dev</th>
<th>Significant (5%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

% @course_averages.each do |row| %
% if row[6] %
<tr style="color: #e7141e">% tr style="color: #090207">% end %
<td style="padding:0 15px 0 15px;" align="left">% row[0] %</td>
<td style="padding:0 15px 0 15px;" align="center">% row[1] %</td>
<td style="padding:0 15px 0 15px;" align="center">% row[2] %</td>
STUDENT LEARNING OUTCOMES SUMMARY

<table>
<thead>
<tr>
<th>SLO</th>
<th>Current number questions answered</th>
<th>Current average</th>
<th>Number Past Terms</th>
<th>Past average</th>
</tr>
</thead>
</table>

BREAKING DOWN SIGNIFICANTLY DIFFERENT RESULTS

Courses
<h2>Student Learning Objectives</h2>

There are no significantly different SLOs

<table>
<thead>
<tr>
<th>Knowledge Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct Answers</td>
</tr>
<tr>
<td>Total Answers</td>
</tr>
<tr>
<td>Percentage</td>
</tr>
</tbody>
</table>

<table border-spacing: 5px align='left'>
<thead align='left'>
<tr>
<th style="padding:0 15px 0 0px;">Knowledge Area</th>
<th style="padding:0 15px 0 15px;">Correct Answers</th>
<th style="padding:0 15px 0 15px;">Total Answers</th>
<th style="padding:0 15px 0 15px;">Percentage</th>
</tr>
</thead>
<tbody>
<tr><td style="padding:0 15px 0 0px;">Question Text</td><td style="padding:0 15px 0 15px;">Correct Answers</td><td style="padding:0 15px 0 15px;">Total Answers</td><td style="padding:0 15px 0 15px;">Percentage</td></tr>
<tr><td style="padding:0 15px 0 0px;">Correct Answers</td><td style="padding:0 15px 0 15px;">Total Answers</td><td style="padding:0 15px 0 15px;">Percentage</td></tr>
</tbody>
</table>
### Knowledge Unit

<table>
<thead>
<tr>
<th>Knowledge Area</th>
<th>Knowledge Unit</th>
<th>Correct Answers</th>
<th>Total Answers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>167</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Knowledge Topic

<table>
<thead>
<tr>
<th>Knowledge Area</th>
<th>Knowledge Unit</th>
<th>Correct Answers</th>
<th>Total Answers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>172</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Knowledge Area</th>
<th>Knowledge Unit</th>
<th>Correct Answers</th>
<th>Total Answers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>184</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Knowledge Area</th>
<th>Knowledge Unit</th>
<th>Correct Answers</th>
<th>Total Answers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>198</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Knowledge Area</th>
<th>Knowledge Unit</th>
<th>Correct Answers</th>
<th>Total Answers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Course</td>
<td>Question Text</td>
<td>Knowledge Area</td>
<td>Knowledge Unit</td>
<td>Knowledge Topic</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

225

<h3>Questions</h3>
<table style='border-spacing: 5px'>
<thead align='left'>
<tr><th style='padding:0 15px 0 0px;' align='left'>Course</th>
<th style='padding:0 15px 0 0px;' align='left'>Question Text</th>
<th style='padding:0 15px 0 0px;' align='left'>Knowledge Area</th>
<th style='padding:0 15px 0 0px;' align='left'>Knowledge Unit</th>
<th style='padding:0 15px 0 0px;' align='left'>Knowledge Topic</th>
<th style='padding:0 15px 0 0px;' align='center'>Correct Answers</th>
<th style='padding:0 15px 0 0px;' align='center'>Total Answers</th>
<th style='padding:0 15px 0 0px;' align='center'>Percentage</th></tr>
</thead>
<tbody>
<tr><td style='padding:0 15px 0 0px;'><%= row[0] %></td><td style='padding:0 15px 0 0px;'><%= row[1] %></td><td style='padding:0 15px 0 0px;'><%= row[2] %></td><td style='padding:0 15px 0 0px;'><%= row[3] %></td><td style='padding:0 15px 0 0px;'><%= row[4] %></td><td style='padding:0 15px 0 0px;'><%= (row[3].to_f / row[4] * 100).round(1) %></td><td style='padding:0 15px 0 0px;'>257</td></tr>
</tbody>
</table>

220

240

258

259

260
CAPTION: A comparison of how many assessment questions are answered correctly versus incorrectly during the Fall, Spring, and Summer terms of the 2014 school year. Each bar represents a student learning outcome for a given semester (number of questions answered correctly to number answered incorrectly).

This chart shows the results of the assessment questions mapped to SLOs including the number of questions answered. The next chart shows similar data but presents it as percentage answered correctly based over term. By continuing to collect this data, we are able to watch for correlation such as impact of class size on student performance.
Both charts are an example of the data that is collected for the self-study under the 17 new system.

In addition to being able to look at the Student Learning Outcomes as a collective, this new assessment system makes it easier to break down the manner in which the data can be explored. It can be broken down based on courses, demographics, course preparedness, final grades, and student learning outcomes. Once broken down it can provide data providing feedback on the success or failure of changes to curriculum such as pathway options and teaching styles. The charts below are provided to show just some of the things that the new data collection system can examine. Note that some of the visualizations are labeled “demonstration only” as data may take several semesters of assessment before delivering statistically significant results.
name: 'Summer 2014 correct',
data: <%= @slos_combined_semesters.map{|a | a[3]} %>,
stack: 'summer2014'
}, {
name: 'Summer 2014 incorrect',
data: <%= @slos_combined_semesters.map{|a | a[4]} %>,
stack: 'summer2014'
}, {
name: 'Fall 2014 correct',
data: <%= @slos_combined_semesters.map{|a | a[5]} %>,
stack: 'fall2014'
}, {
name: 'Fall 2014 incorrect',
data: <%= @slos_combined_semesters.map{|a | a[6]} %>,
stack: 'fall2014'
});
});
</script>
<div id="container" style="min-width: 310px; height: 400px; margin: 0 auto"></div>

<h2>SLO combine table</h2>
<br>
<script type="text/javascript" charset="UTF-8">
$(function () {
  $('#lineChart').highcharts({
title: {
text: 'SLO Assessment 2014',
x: -20 //center
},
subtitle: {
text: 'Source: WorldClimate.com',
x: -20
},
xAxis: {
categories: ['Spring', 'summer', 'fall']
},
yAxis: {
title: {
text: 'Questions answered correctly (%)'
},
plotLines: [{
value: 0,
width: 1,
color: '#808080'
}]
},
tooltip: {
valueSuffix: '%'
},
legend: {
layout: 'vertical',
align: 'right',
verticalAlign: 'middle',
borderWidth: 0
},
series: [{
name: 'a',
data: <%= @slos_percentages[0][1..-1] %>
},
{name: 'b',
data: <%= @slos_percentages[1][1..-1] %>
},
{name: 'c',
data: <%= @slos_percentages[2][1..-1] %>
},
{name: 'h',
data: <%= @slos_percentages[3][1..-1] %>
},
{name: 'i',
data: <%= @slos_percentages[4][1..-1] %>
}];
});
</script>
A different view of the comparison made in the above chart on how many assessment questions are answered correctly versus incorrectly during the Fall, Spring, and Summer terms of the 2014 school year. Here percentage answered correctly is used.

Within our program, there are several different paths that a student could take (See Figure 4). It is assumed that the same knowledge is obtained from all the courses, but with the new assessment system, it can be easily monitored by performance on pre-assessment exams. In the next data sets the student results are broken down by pathways: those who took CS 1160/1, and those who did not (they either started in CS 1180 or transferred).
series: [{
  name: '1160 Students A',
  data: <%= @grades_for_graph.map{|a| a[1]} %>,
  stack: 'students1160'
},
  name: '1160 Students B',
  data: <%= @grades_for_graph.map{|a| a[2]} %>,
  stack: 'students1160'
},
  name: '1160 Students C',
  data: <%= @grades_for_graph.map{|a| a[3]} %>,
  stack: 'students1160'
},
  name: '1160 Students D',
  data: <%= @grades_for_graph.map{|a| a[4]} %>,
  stack: 'students1160'
},
  name: '1160 Students F',
  data: <%= @grades_for_graph.map{|a| a[5]} %>,
  stack: 'students1160'
},
  name: 'Other Students A',
  data: <%= @grades_for_graph.map{|a| a[6]} %>,
  stack: 'students1180'
},
  name: 'Other Students B',
  data: <%= @grades_for_graph.map{|a| a[7]} %>,
  stack: 'students1180'
},
  name: 'Other Students C',
  data: <%= @grades_for_graph.map{|a| a[8]} %>,
  stack: 'students1180'
},
  name: 'Other Students D',
  data: <%= @grades_for_graph.map{|a| a[9]} %>,
  stack: 'students1180'
},
  name: 'Other Students F',
  data: <%= @grades_for_graph.map{|a| a[10]} %>,
  stack: 'students1180'
}]);
});
</script>

<svg id="gradesChart" style="min-width: 310px; height: 400px; margin: 0 auto"></svg>

<h2>Students in 1160</h2>
<table style="width:50%">
<thead>
<tr>
<th>Course</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
<th></th>
</tr>
</thead>
</table>
<p>The next chart looks simply at how many students took the course. This allows us to observe if there is a significant drop off of students from one of the pathways at some point through the core sequence.</p>
The next data set is similar to the previous but it looks compares students who took CS 1200 and(or) CS 2200 versus those students who took MTH 2570. Both these pathways look at Discrete structures but the CS 1200/2200 sequence is designed for under prepared students.
<div id="gradesDiscreteChart" style="min-width: 310px; height: 400px; margin: 0 auto"></div>

<h2>Students in MTH 2570</h2>
<table style="width:50%">
<thead>
<tr>
<th>Course</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr><td>CS 1200 and(or) CS 2200 Students A'</td> 491 492
<tr><td>CS 1200 and(or) CS 2200 Students B'</td> 503
<tr><td>CS 1200 and(or) CS 2200 Students C'</td> 507
<tr><td>CS 1200 and(or) CS 2200 Students D'</td> 517
<tr><td>CS 1200 and(or) CS 2200 Students F'</td> 522
</tbody>
</table>

<h2>Students in CS 1200 and(or) CS 2200</h2>
<table style="width:50%">
<thead>
<tr>
<th>Course</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr><td>CS 1200 and(or) CS 2200 Students A'</td> 491 492
<tr><td>CS 1200 and(or) CS 2200 Students B'</td> 503
<tr><td>CS 1200 and(or) CS 2200 Students C'</td> 507
<tr><td>CS 1200 and(or) CS 2200 Students D'</td> 517
<tr><td>CS 1200 and(or) CS 2200 Students F'</td> 522
</tbody>
</table>
The next chart looks simply at how many students took the course. This allows us to observe if there is a significant drop off of students from one of the pathways at some point through the core sequence.

What follows is a breakdown of student answers by both knowledge topic and then question. While this is a little more cumbersome, it allows for specific problems to be identified.
Knowledge Topic Totals

Question Totals

Flexibility with the New Assessment System

It is important to note that anything that is based on pre-assessment grade can easily be linked to student learning outcomes and program educational objectives since all the questions are mapped to what they test. Similarly, anything that shows student learning outcomes or program educational objectives can be broken down into specific knowledge topics, questions, and students who answered those questions. Since all the results are linked back to a specific student, all the questions can be broken down by demographics, year began, pathways, or even previous performance in a course or on a pre-assessment quiz. This makes the items displayed in this paper easy to mix and match. Also note that while charts are largely used in this paper for ease of viewing, all the values in the charts have hard numbers backing them up. Statistical tests can be run on areas of interest to determine if a change really occurred before any action is taken. Additionally, the results can be presented as exact sample size or as percentages.

This assessment infrastructure allows for an assessment of retained knowledge, topic by topic, for each individual student, course, and term. When collected with appropriate demographic information, these assessments allow the differential measurements of knowledge retention under any number of pedagogical variables. The success of new instructional styles, laboratory techniques, or technologies for developing knowledge can be assessed against different approaches.

Every contemporary engineering discipline has a professional society that helps identify the core concepts of the discipline. Indeed, most engineering disciplines have standardized examinations of some sort that are used to demonstrate student proficiency for licensure or graduate studies. Questions of this sort can be used at the start of core courses or time points to assess student knowledge of prerequisite topics developed earlier in any program of study. These assessments can be delivered as online questions to minimize cost and maximize participation. When collected
with appropriate demographic information, this rich set of data can guide program improvement more effectively than many existing program assessment plans. Although we present this infrastructure in the context of Computer Science, we believe that the approach can be applied to implement an infrastructure for effective assessment program for any engineering discipline.

Bibliography


This Data is not accurate!!!
<td style="padding:0 15px 0 15px;" align="center"><% = student.gender %></td>
<td style="padding:0 15px 0 15px;" align="center"><% = row.grade %></td></tr>
<% end %></tbody></table>

34 35 36 <br><br><%= link_to 'Home Page', root_path %>

File - D:\railsApps\playgroundJan2016\app\views\standard_reports\results_by_semester.html.erb

30</table>

File - D:\railsApps\playgroundJan2016\app\views\standard_reports\single_course_report.html.erb

1 <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"
type="text/javascript"></script>

7 <h2>Courses</h2>

10 <%= render partial: "bar_chart_single_course", locals: {function_name: 'course_bar_chart', title: 'CS 1181 Results for Spring Control Spring CS 1180 Students', table: @chart_data} %>

11 <table style='border-spacing: 5px'>
<thead align='left'>
<tr>
<th style='padding:0 15px 0 15px;" align="left">Question Text</th>
<th style='padding:0 15px 0 15px;" align="center">Correct Answers</th>
</tr>
</thead>
</table>
<table>
<thead>
<tr>
<th>Student</th>
<th>question</th>
<th>correct</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>50%</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>25%</td>
</tr>
</tbody>
</table>

COURSE ANSWERS

1. A
2. B
3. C
4. D
5. E
6. F
7. G
8. H
9. I
10. J
11. K
12. L
13. M
14. N
15. O
16. P
17. Q
18. R
19. S
20. T
21. U
22. V
23. W
24. X
25. Y
26. Z
27. AA
28. AB
29. AC
30. AD
31. AE
32. AF
33. AG
34. AH
35. AI

File - D:\railsApps\playgroundJan2016\app\views\standard_reports\quiz_results_by_class.html.erb

<script src="https://code.highcharts.com/modules/exporting.js"></script>

6
<br>
<br>
<h1>COURSE ANSWERS</h1>

10
11
<table style='border-spacing: 5px'>
<thead>
<tr>
<th style="padding:0 15px 0 0px;" align="left">Student</th>
<th style="padding:0 15px 0 15px;" align="center">question</th>
<th style="padding:0 15px 0 15px;" align="center">correct</th>
</tr>
</thead>
<tbody>
<tr><td style="padding:0 15px 0 0px;" align="left">Student.find(row.w_number).w_number</td>
<td style="padding:0 15px 0 15px;" align="center">row.question_text</td>
<td style="padding:0 15px 0 15px;" align="center">row.correct</td>
</tr>
</tbody>
</table>

32
33
34
<br>
<br><%= link_to 'Home Page', root_path %>
<table style="width:75%">
<thead>
<th>Title</th>
<th>Correct</th>
<th>Incorrect</th>
</thead>
<tbody>
<tr>
<td><%= row[0] %></td>
<td><%= row[1] %></td>
<td><%= row[2] %></td>
</tr>
</tbody>
</table>
plotOptions: {
  series: {
    stacking: 'normal'
  }
},
series: [
  {% i = 1 %
  {% while i < (table.length - 3) %}
  {
    name: '<%= raw table[table.length - 1][i - 1]%> correct',
    data: '<%= raw table[i]%>,
    stack: '<%= raw table[table.length - 1][i-1] %>'
  },
  {% i += 1 %}
  {% end %}
  },
  {% i += 1 %}
  {% end %
  {
    name: '<%= raw table.last.last %> correct',
    data: '<%= raw table[table.length-3]%>,
    stack: '<%= raw table[table.length-last] %>'
  },
  {
    name: '<%= raw table.last.last%> incorrect',
    data: '<%= raw table[table.length-last]%>,
    stack: '<%= raw table.last.last%>'
  }]
};

</script>
<div id='<<%= title %>'></div>

<% @cac_slos.each do |slo| %>
  <h2>CAC: <%= slo %></h2>
  <table>
    <thead align='left'>
      <tr style="min-width: 100px;padding:0 15px 0 15px;align="left">Course</tr>
    </thead>
    <tbody>
      <tr>
        <td><%= slo %></td>
      </tr>
    </tbody>
  </table>
</%>
<th style="min-width: 350px; padding: 0 15px 0 15px" align="left">Knowledge Topic</th>

@terms.each do |term|  
<th style="min-width: 80px; padding: 0 15px 0 15px" align="center"><%= term %></th>

@slos_with_course_and_kt.each do |row|  
<tr>
<td style="padding: 0 15px 0 0px" align="left"><%= row[0] %> </td>
<td style="padding: 0 15px 0 0px" align="left"><%= row[1].concat('/'.concat(row[2].concat(': '.concat(row[3]))) %></td>
<td style="padding: 0 15px 0 0px" align="center">  
<% for i in 5..19 %>
<% if row[i][0] == 0 and row[i][1] == 0 %>
<td style="padding: 0 15px 0 0px" align="center">
</td>
<% else %>
<td style="padding: 0 15px 0 0px" align="center"><%= row[i][0] %> <%= row[i][1] %></td>
<% end %>
</tr>
<% end %>
</tbody>
</table>

EAC: <%= slo %>

@slos_with_course_and_kt.each do |row|  
<tr>
<td style="padding: 0 15px 0 0px" align="left"><%= row[0] %> </td>
<td style="padding: 0 15px 0 0px" align="left"><%= row[1].concat('/'.concat(row[2].concat(': '.concat(row[3]))) %></td>
<td style="padding: 0 15px 0 0px" align="center">  
<% for i in 5..19 %>
<% if row[i][0] == 0 and row[i][1] == 0 %>
<td style="padding: 0 15px 0 0px" align="center">
</td>
<% else %>
<td style="padding: 0 15px 0 0px" align="center"><%= row[i][0] %> <%= row[i][1] %></td>
<% end %>
</tr>
<% end %>
</tbody>
</table>

EAC: <%= slo %>

@slos_with_course_and_kt.each do |row|  
<tr>
<td style="padding: 0 15px 0 0px" align="left"><%= row[0] %> </td>
<td style="padding: 0 15px 0 0px" align="left"><%= row[1].concat('/'.concat(row[2].concat(': '.concat(row[3]))) %></td>
<td style="padding: 0 15px 0 0px" align="center">  
<% for i in 5..19 %>
<% if row[i][0] == 0 and row[i][1] == 0 %>
<td style="padding: 0 15px 0 0px" align="center">
</td>
<% else %>
<td style="padding: 0 15px 0 0px" align="center"><%= row[i][0] %> <%= row[i][1] %></td>
<% end %>
</tr>
<% end %>
</tbody>
</table>

EAC: <%= slo %>

@slos_with_course_and_kt.each do |row|  
<tr>
<td style="padding: 0 15px 0 0px" align="left"><%= row[0] %> </td>
<td style="padding: 0 15px 0 0px" align="left"><%= row[1].concat('/'.concat(row[2].concat(': '.concat(row[3]))) %></td>
<td style="padding: 0 15px 0 0px" align="center">  
<% for i in 5..19 %>
<% if row[i][0] == 0 and row[i][1] == 0 %>
<td style="padding: 0 15px 0 0px" align="center">
</td>
<% else %>
<td style="padding: 0 15px 0 0px" align="center"><%= row[i][0] %> <%= row[i][1] %></td>
<% end %>
</tr>
<% end %>
</tbody>
</table>

EAC: <%= slo %>

@slos_with_course_and_kt.each do |row|  
<tr>
<td style="padding: 0 15px 0 0px" align="left"><%= row[0] %> </td>
<td style="padding: 0 15px 0 0px" align="left"><%= row[1].concat('/'.concat(row[2].concat(': '.concat(row[3]))) %></td>
<td style="padding: 0 15px 0 0px" align="center">  
<% for i in 5..19 %>
<% if row[i][0] == 0 and row[i][1] == 0 %>
<td style="padding: 0 15px 0 0px" align="center">
</td>
<% else %>
<td style="padding: 0 15px 0 0px" align="center"><%= row[i][0] %> <%= row[i][1] %></td>
<% end %>
</tr>
<% end %>
</tbody>
</table>

EAC: <%= slo %>

@slos_with_course_and_kt.each do |row|  
<tr>
<td style="padding: 0 15px 0 0px" align="left"><%= row[0] %> </td>
<td style="padding: 0 15px 0 0px" align="left"><%= row[1].concat('/'.concat(row[2].concat(': '.concat(row[3]))) %></td>
<td style="padding: 0 15px 0 0px" align="center">  
<% for i in 5..19 %>
<% if row[i][0] == 0 and row[i][1] == 0 %>
<td style="padding: 0 15px 0 0px" align="center">
</td>
<% else %>
<td style="padding: 0 15px 0 0px" align="center"><%= row[i][0] %> <%= row[i][1] %></td>
<% end %>
</tr>
<% end %>
</tbody>
</table>

EAC: <%= slo %>
### ABET SUMMARY TABLE

<table>
<thead>
<tr>
<th>Knowledge Topic</th>
<th>SLO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```ruby
foreach i in 5..19 do
  if row[i][0] == 0 and row[i][1] == 0
    td style="padding:0 15px 0 0px" align="center">
  else
    td style="padding:0 15px 0 0px" align="center">
end
```
<td style="padding:0 15px 0 0px" align="center">[<%= row[i][0] %>, <%= row[i][1] %>]</td>
</tr>
</tbody></table>
46
47
File -  
D:\railsApps\playgroundJan2016\app\views\program_of_studies\new.html.erb
1  <h1>New Program Of Study</h1>
2
3  <%= render 'form' %>
4
5  <%= link_to 'Back', program_of_studies_path %>
File - D:\railsApps\playgroundJan2016\app\views\program_of_studies\edit.html.erb
<pre>1  &lt;h1&gt;Editing Program Of Study&lt;/h1&gt;
2 3 &lt;%= render 'form' %&gt;
4 &lt;%= link_to 'Show', @program_of_study %&gt; | &lt;%= link_to 'Back', program_of_studies_path %&gt; 7
</pre>

File - D:\railsApps\playgroundJan2016\app\views\program_of_studies\show.html.erb
<pre>1 &lt;p id="notice"&gt;&lt;%= notice %&gt;&lt;/p&gt;
2 &lt;p&gt;
3 &lt;strong&gt;Program:&lt;/strong&gt; 4 &lt;%= @program_of_study.program %&gt; 
5 &lt;/p&gt; 7
6 &lt;p&gt;
7 &lt;strong&gt;Enroll:&lt;/strong&gt; 8 &lt;%= @program_of_study.Enroll %&gt; &lt;/p&gt;
12 &lt;%= link_to 'Edit', edit_program_of_study_path(@program_of_study) %&gt; | &lt;%= link_to 'Back', program_of_studies_path %&gt; 15
</pre>

File - D:\railsApps\playgroundJan2016\app\views\program_of_studies\_form.html.erb
<pre>&lt;% if @program_of_study.errors.any? %&gt;
&lt;div id="error_explanation"&gt;
 &lt;h2&gt;&lt;%= pluralize(@program_of_study.errors.count, "error") %&gt;&lt;/h2&gt;
5 &lt;ul&gt;&lt;%= @program_of_study.errors.full_messages.each do |message| %&gt;&lt;li&gt;&lt;%= message %&gt;&lt;/li&gt;&lt;/% end %&gt;&lt;/ul&gt;&lt;/div&gt;&lt;% end %&gt; 13
&lt;div class="field"&gt;
 &lt;%= f.label :program %&gt;&lt;br&gt;&lt;%= f.text_field :program %&gt;&lt;/div&gt;
&lt;div class="field"&gt;
 &lt;%= f.label :Enroll_id %&gt;&lt;br&gt;&lt;%= f.text_field :Enroll_id %&gt;&lt;/div&gt;
&lt;div class="actions"&gt;
 &lt;%= f.submit %&gt;
&lt;/div&gt;&lt;% end %&gt; 26
</pre>

File - D:\railsApps\playgroundJan2016\app\views\program_of_studies\index.html.erb
<pre>1 &lt;p id="notice"&gt;&lt;%= notice %&gt;&lt;/p&gt;
2 &lt;h1&gt;Listing Program Of Studies&lt;/h1&gt; 4
 &lt;table&gt;
 &lt;thead&gt;
 &lt;tr&gt;
 &lt;th&gt;Program&lt;/th&gt; 239
<th><strong>Enroll</strong></th>
<th colspan="3"></th>
</tr>
</thead>

13
<tbody>
<% @program_of_studies.each do |program_of_study| %>
<tr>
<td><%= program_of_study.program %></td>
<td><%= program_of_study.Enroll %></td>
<td><%= link_to 'Show', program_of_study %></td>
<td><%= link_to 'Edit', edit_program_of_study_path(program_of_study) %></td>
<td><%= link_to 'Destroy', program_of_study, method: :delete, data: { confirm: 'Are you sure?' } %></td>
</tr>
<% end %>
</tbody>
</table>

26
27<br>
28
29 <%= link_to 'New Program of study', new_program_of_study_path %> 30

json.extract! @program_of_study, :id, :program, :Enroll_id
json.url program_of_study_url(program_of_study, format: :json)
end

5
File D:\railsApps\playgroundJan2016\app\views\slo_covered_by_kts\new.html.erb
1 <h1>New Slo Covered By Kt</h1>
2 <%= render 'form' %>
3 <%= link_to 'Back', slo_covered_by_kts_path %>

File D:\railsApps\playgroundJan2016\app\views\slo_covered_by_kts\edit.html.erb
1 <h1>Editing Slo Covered By Kt</h1>
2 <%= render 'form' %>
3 <%= link_to 'Show', @slo_covered_by_kt %>
4 <%= link_to 'Back', slo_covered_by_kts_path %>

File D:\railsApps\playgroundJan2016\app\views\slo_covered_by_kts\show.html.erb
1 <p id="notice"><%= notice %></p>
2 <p><strong>Studentlearningoutcome:</strong></p>
3 <%= @slo_covered_by_kt.StudentLearningOutcome %>
4 <p><strong>Knowledgetopic:</strong></p>
5 <%= link_to 'Edit', edit_slo_covered_by_kt_path(@slo_covered_by_kt) %>
6 <%= link_to 'Back', slo_covered_by_kts_path %>

File D:\railsApps\playgroundJan2016\app\views\slo_covered_by_kts\form.html.erb
13 <div class="field">
14 <%= f.label :StudentLearningOutcome_id %><br>
15 <%= f.text_field :StudentLearningOutcome_id %>
</div>
16 <div class="field">
17 <%= f.label :KnowledgeTopic %><br>
18 <%= f.text_field :KnowledgeTopic %>
</div>
<%= f.label :KnowledgeTopic_id %><br>
<%= f.text_field :KnowledgeTopic_id %>
</div>
<div class="actions">
<%= f.submit %>
</div>
</div>

<% end %>
26
File  D:\railsApps\playgroundJan2016\app\views\slo_covered_by_kts\index.html.erb
1 <p id="notice"><%= notice %></p>
2
3 <h1>Listing Slo Covered By Kts</h1>
4
<table><thead><tr><th>Studentlearningoutcome</th>
th<KnowledgeTopic</th>
th colspan="5"></tr></thead><tbody><% @slo_covered_by_kts.each do |slo_covered_by_kt| %><tr><td><%= (slo_covered_by_kt.student_learning_outcome_id) %></td><td><%= StudentLearningOutcome.find(slo_covered_by_kt.student_learning_outcome_id).accredidation_body %></td><td><%= slo_covered_by_kt.knowledge_topic_id %></td><td><%= KnowledgeTopic.find(slo_covered_by_kt.knowledge_topic_id).knowledge_topic %></td><td><%= link_to 'Show', slo_covered_by_kt %></td><td><%= link_to 'Edit', edit_slo_covered_by_kt_path(slo_covered_by_kt) %></td><td><%= link_to 'Destroy', slo_covered_by_kt, method: :delete, data: { confirm: 'Are you sure?' } %></td></tr><% end %></tbody></table>

28
29 <br>
30
31 <%= link_to 'New Slo covered by kt', new_slo_covered_by_kt_path %>
32
File  D:\railsApps\playgroundJan2016\app\views\slo_covered_by_kts\show.json.jbuilder
1 json.extract! @slo_covered_by_kt, :id, :StudentLearningOutcome_id, :KnowledgeTopic_id, :created_at, :updated_at
2
File  D:\railsApps\playgroundJan2016\app\views\slo_covered_by_kts\index.json.jbuilder
json.array!(@slo_covered_by_kts) do |slo_covered_by_kt| json.extract! slo_covered_by_kt, :id, :StudentLearningOutcome_id, :KnowledgeTopic_id json.url slo_covered_by_kt_url(slo_covered_by_kt, format: :json) end
<h1>New Peo Covered By Slo</h1>

<%= render 'form' %>

<%= link_to 'Back', peo_covered_by_slos_path %>

<h1>Editing Peo Covered By Slo</h1>

<%= render 'form' %>

<%= link_to 'Show', @peo_covered_by_slo %>
| <%= link_to 'Back', peo_covered_by_slos_path %>

<p id="notice"><%= notice %></p>

<p><strong>Student Learning Outcome:</strong><%= @peo_covered_by_slo.StudentLearningOutcome %></p>

<p><strong>Program Educational Objective:</strong><%= @peo_covered_by_slo.ProgramEducationalObjective %></p>

<p><strong>Program Educational Objective:</strong><%= @peo_covered_by_slo.ProgramEducationalObjective %></p>

<%= link_to 'Edit', edit_peo_covered_by_slo_path(@peo_covered_by_slo) %>
| <%= link_to 'Back', peo_covered_by_slos_path %>

<p id="error_explanation">
<h2><%= pluralize(@peo_covered_by_slo.errors.count, "error") %>
 prohibted this peo_covered_by_slo from being saved:</h2>
<ul>
  <% @peo_covered_by_slo.errors.full_messages.each do |message| %>
  <li><%= message %></li>
  <% end %>
</ul>
</div>

<%= form_for(@peo_covered_by_slo) do |f| %>

  <div id="error_explanation">
    <h2><%= pluralize(@peo_covered_by_slo.errors.count, "error") %>
    prohibited this peo_covered_by_slo from being saved:</h2>
    <ul>
      <% @peo_covered_by_slo.errors.full_messages.each do |message| %>
      <li><%= message %></li>
      <% end %>
    </ul>
  </div>

  <%= f.label :StudentLearningOutcome_id %>
  <%= f.text_field :StudentLearningOutcome_id %>
</div>

<%= render 'form' %>

<%= link_to 'Back', peo_covered_by_slos_path %>
Listing Peo Covered By Slos

<table>
<thead>
<tr>
<th>StudentLearningOutcome</th>
<th>ProgramEducationalObjective</th>
<th>Show</th>
<th>Edit</th>
<th>Destroy</th>
</tr>
</thead>
<tbody>
<tr>
<td>peo_covered_by_slo</td>
<td>programEducationalObjective</td>
<td>link_to</td>
<td>link_to</td>
<td>link_to</td>
</tr>
</tbody>
</table>

New Peo covered by slo
File
D:\railsApps\playgroundJan2016\app\views\load_internal_reports\index.html.erb
1
<h2>Upload IR Report:</h2>
<% form_tag "/load_internal_reports/load_report", multipart: :true do %>
<%= file_field_tag :file %>
<br><br>
<%= submit_tag "Load IR report" %>
<% end %>
Views: KTS Covered By Answers

File - D:\railsApps\playgroundJan2016\app\views\kts_covered_by_answers\new.html.erb
1  <h1>New Kts Covered By Answer</h1>
2  3 <%= render 'form' %>
4  5 <%= link_to 'Back', kts_covered_by_answers_path %>

File - D:\railsApps\playgroundJan2016\app\views\kts_covered_by_answers\edit.html.erb
1  <h1>Editing Kts Covered By Answer</h1>
2  3 <%= render 'form' %>
4  5 <%= link_to 'Show', @kts_covered_by_answer %>
6  7 <%= link_to 'Back', kts_covered_by_answers_path %>

File - D:\railsApps\playgroundJan2016\app\views\kts_covered_by_answers\_form.html.erb
1  <%= form_for(@kts_covered_by_answer) do |f| %>
2  3 <% if @kts_covered_by_answer.errors.any? %>
3  4 <div id="error_explanation">
4  5 <h2><%= pluralize(@kts_covered_by_answer.errors.count, "error") %>
5  6 prohibited this kts_covered_by_answer from being saved:</h2>
5  7 <ul>
6  8 <% @kts_covered_by_answer.errors.full_messages.each do |message| %>
6  9 10 <li><%= message %></li>
6  11 12 end %>
6  13 </ul>
5  14 </div>
5  15 <div class="field">
5  16 <%= f.label :knowledge_topic_id %>
5  17 <%= f.text_field :knowledge_topic_id %>
5  18 </div>

<% strong>Answer:</strong>
<% @kts_covered_by_answer.answer %>
</p>
12
<%= link_to 'Edit', edit_kts_covered_by_answer_path(@kts_covered_by_answer) %>
<%= link_to 'Back', kts_covered_by_answers_path %>

File - D:\railsApps\playgroundJan2016\app\views\kts_covered_by_answers\show.html.erb
1  <p id="notice"><%= notice %></p>
2 3 <p><strong>Knowledge topic:</strong></p>
4 5 <%= @kts_covered_by_answer.knowledge_topic %>
6 7 8 <p><strong>Answer:</strong><%= @kts_covered_by_answer.answer %>
9 10 <p>
11 <p>
12 <p>
13 <p>
14 <p>
Listing Kts Covered By Answers

<table>
<thead>
<tr>
<th>Knowledge topic</th>
<th>Answer</th>
<th>Show</th>
<th>Edit</th>
<th>Destroy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge area</td>
<td>Kts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knowledge area</td>
<td>Kts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knowledge area</td>
<td>Kts</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New Kts covered by answer

Show

Edit

Destroy

File - D:\railsApps\playgroundJan2016\app\views\load_slo_covered_by_kts\index.html.erb

<h2>Upload SLO covered by KT:s:</h2>
<%= form_tag "/load_slo_covered_by_kts/handle_form", multipart: :true do %>
<%= file_field_tag :file %>
<br><br>
<%= submit_tag "Load SLOs covered By KT:s" %>
<% end %>

File
D:\railsApps\playgroundJan2016\app\views\load_student_quiz_results\index.html.erb
<br>
<h1>Upload Quiz Results</h1>
3
4
<h3>Upload quiz results by section:</h3>

```erb
<%= form_tag "/load_student_quiz_results/load_quiz", multipart: true do %>
<%= file_field_tag :file %>
<br>
<%= label_tag(:crn, "CRN:" ) %>
<%= text_field_tag(:crn) %>
<br>
<%= label_tag(:course, "Program (CS or CEG):") %>
<%= text_field_tag(:course) %>
<br>
<%= label_tag(:courseNumber, "Course Number:" ) %>
<%= text_field_tag(:courseNumber) %>
<br>
<%= label_tag(:instrLastName, "Faculty Last Name:" ) %>
<%= text_field_tag(:instrLastName) %>
<br>
<%= label_tag(:instrFirstName, "Faculty First Name:" ) %>
<%= text_field_tag(:instrFirstName) %>
<br>
<%= submit_tag "Import" %>
<% end %>
```

File
D:\railsApps\playgroundJan2016\app\views\student_learning_outcomes\new.html.erb
1 <h1>New Student Learning Outcome</h1>
2
3 <%= render 'form' %>
4
5 <%= link_to 'Back', student_learning_outcomes_path %>

file - D:\railsApps\playgroundJan2016\ app\views\student_learning_outcomes\edit.html.erb
1 <h1>Editing Student Learning Outcome</h1>
2 3 <%= render 'form' %>
4 <%= link_to 'Show', @student_learning_outcome %> |
5 <%= link_to 'Back', student_learning_outcomes_path %>

File - D:\railsApps\playgroundJan2016\ app\views\student_learning_outcomes\show.html.erb
1 <p id="notice"><%= notice %></p>
2 3 <p><strong>Accredidation body:</strong> <%= @student_learning_outcome.accredidation_body %></p>
4 7 8 <p><strong>Title:</strong> <%= @student_learning_outcome.title %></p>
9 12 13 <p><strong>Description:</strong> <%= @student_learning_outcome.description %></p>
14 17 18 <p><strong>Year added:</strong> <%= @student_learning_outcome.year_added %></p>
19 22 23 <p><strong>Active:</strong> <%= @student_learning_outcome.active %></p>
24 27 28 <p><strong>Correct answers:</strong> <%= @student_learning_outcome.correct_answers %></p>
29 32 33 <p><strong>Incorrect answers:</strong> <%= @student_learning_outcome.incorrect_answers %></p>
34 37 38 <p><strong>Temp correct answer:</strong> <%= @student_learning_outcome.temp_correct_answer %></p>
39 42 43 <p><strong>Temp incorrect answer:</strong> <%= @student_learning_outcome.temp_incorrect_answer %></p>
44 47 48 <p><%= link_to 'Edit', edit_student_learning_outcome_p
<% if @student_learning_outcome.errors.any? %>
  <div id="error_explanation">
    <h2>
      Prohibited this student_learning_outcome from being saved:
    </h2>
    <ul>
      <% @student_learning_outcome.errors.full_messages.each do |message| %>
        <li><%= message %></li>
      <% end %>
    </ul>
  </div>
<% end %>

<div class="field">
  <%= f.label :accredidation_body %>
  <%= f.text_field :accredidation_body %>
</div>

<div class="field">
  <%= f.label :title %>
  <%= f.text_field :title %>
</div>

<div class="field">
  <%= f.label :description %>
  <%= f.text_field :description %>
</div>

<div class="field">
  <%= f.label :year_added %>
  <%= f.number_field :year_added %>
</div>

<div class="field">
  <%= f.label :active %>
  <%= f.check_box :active %>
</div>

<div class="field">
  <%= f.label :correct_answers %>
  <%= f.number_field :correct_answers %>
</div>

<div class="field">
  <%= f.label :incorrect_answers %>
  <%= f.number_field :incorrect_answers %>
</div>

<div class="field">
  <%= f.label :temp_correct_answer %>
  <%= f.number_field :temp_correct_answer %>
</div>

<div class="field">
  <%= f.label :temp_incorrect_answer %>
  <%= f.number_field :temp_incorrect_answer %>
</div>

</div>
**Listing Student Learning Outcomes**

<table>
<thead>
<tr>
<th>Accredidation Body</th>
<th>Title</th>
<th>Description</th>
<th>Year added</th>
<th>Correct answers</th>
<th>Incorrect answers</th>
<th>Temp correct answer</th>
<th>Temp incorrect answer</th>
<th>Show</th>
<th>Edit</th>
<th>Destroy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accredidation Body</td>
<td>Title</td>
<td>Description</td>
<td>Year added</td>
<td>Correct answers</td>
<td>Incorrect answers</td>
<td>Temp correct answer</td>
<td>Temp incorrect answer</td>
<td>Show</td>
<td>Edit</td>
<td>Destroy</td>
</tr>
</tbody>
</table>

- `<table>`
- `<tr>`
- `<th>` Accredidation Body
- `<th>` Title
- `<th>` Description
- `<th>` Year added
- `<th>` Correct answers
- `<th>` Incorrect answers
- `<th>` Temp correct answer
- `<th>` Temp incorrect answer
- `<tr>`
- `<td>` Accredidation Body
- `<td>` Title
- `<td>` Description
- `<td>` Year added
- `<td>` Correct answers
- `<td>` Incorrect answers
- `<td>` Temp correct answer
- `<td>` Temp incorrect answer
- `<td>` Show
- `<td>` Edit
- `<td>` Destroy
File
D:\railsApps\playgroundJan2016\app\views\load_student_learning_outcomes\index.html.erb

<h1>LoadStudentLearningOutcomes</h1>

<h2>Upload SLOs:</h2>

<%= form_tag
"/load_student_learning_outcomes/handle_form", multipart: true do %>
<%= file_field_tag :file %>

<br>

<%= submit_tag "Load SLO" %>
<% end %>


File  
D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\new.html.erb  
1   <h1>New Program Educational Objective</h1>  
2 3 <%= render 'form' %>  
4 5 <%= link_to 'Back', program_educational_objectives_path %>  

File  
D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\edit.html.erb  
1   <h1>Editing Program Educational Objective</h1>  
2 3 <%= render 'form' %>  
4 <%= link_to 'Show', @program_educational_objective %>  
5 <%= link_to 'Back', program_educational_objectives_path %>  

File  
D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\show.html.erb  
1 <p id="notice"><%= notice %></p>  
2  
<p><strong>Accreditation body:</strong><br/>  
%= @program_educational_objective.accredidation_body %></p>  

File  
D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\show.html.erb  
1 <p><strong>Title:</strong><br/>  
%= @program_educational_objective.title %></p>  
2 12 <p><strong>Description:</strong><br/>  
%= @program_educational_objective.description %></p>  
17 18 <p><strong>Year added:</strong><br/>  
%= @program_educational_objective.year_added %></p>  
22 23 <p><strong>Active:</strong><br/>  
%= @program_educational_objective.active %></p>  
27 28 <p><strong>Correct answers:</strong><br/>  
%= @program_educational_objective.correct_answers %></p>  
32 33 <p><strong>Incorrect answers:</strong><br/>  
%= @program_educational_objective.incorrect_answers %></p>  
37 38 <p>
Temp correct answer:

```erb
<%= @program_educational_objective.temp_correct_answer %>
</p>
42
</p>
```

Temp incorrect answer:

```erb
<%= @program_educational_objective.temp_incorrect_answer %>
</p>
47
</p>
```

<table>
<thead>
<tr>
<th>Edit</th>
<th>Back</th>
</tr>
</thead>
</table>

File

D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\_form.html.erb

```erb
<% form_for(@program_educational_objective) do |f| %>
  <% if @program_educational_objective.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@program_educational_objective.errors.count, "error") %>
        prohibited this program_educational_objective from being saved:</h2>
      <ul>
        <%= @program_educational_objective.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :accredidation_body %><br>
    <%= f.text_field :accredidation_body %>
  </div>

  <div class="field">
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </div>

  <div class="field">
    <%= f.label :description %><br>
    <%= f.text_field :description %>
  </div>

  <div class="field">
    <%= f.label :year_added %><br>
    <%= f.number_field :year_added %>
  </div>

  <div class="field">
    <%= f.label :active %><br>
    <%= f.check_box :active %>
  </div>

  <div class="field">
    <%= f.label :correct_answers %><br>
    <%= f.number_field :correct_answers %>
  </div>

  <div class="field">
    <%= f.label :incorrect_answers %><br>
    <%= f.number_field :incorrect_answers %>
  </div>
```

```
Listing Program Educational Objectives

<table>
<thead>
<tr>
<th>Accredidation body</th>
<th>Title</th>
<th>Description</th>
<th>Year added</th>
<th>Active</th>
<th>Correct answers</th>
<th>Incorrect answers</th>
<th>Temp correct answer</th>
<th>Temp incorrect answer</th>
<th>Show</th>
<th>Edit</th>
<th>Destroy</th>
</tr>
</thead>
</table>

Page 1 of 2

File

D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\_form.html.erb

Page 2 of 2

File

D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\index.html.erb
40
41 <br>
42 43 <%= link_to 'New Program educational objective', new_program_educational_objective_path %>

File
- D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\show.json.jbuilder

```json
json.extract!
@program_educational_objective,
```

File
- D:\railsApps\playgroundJan2016\app\views\program_educational_objectives\index.json.jbuilder

```json
json.array!(@program_educational_objectives) do |program_educational_objective|
  json.extract!
  @program_educational_objective,
  json.url
  program_educational_objective_url(program_educational_objective, format: :json)
end
```
body {
  background-color: #fff;
  color: #333;
  font-family: verdana, arial, helvetica, sans-serif;
  font-size: 13px;
  line-height: 18px;
}

p, ol, ul, td {
  font-family: verdana, arial, helvetica, sans-serif;
  font-size: 13px;
  line-height: 18px;
}

pre {
  background-color: #eee;
  padding: 10px;
  font-size: 11px;
}

a {
  color: #000;
}

&:visited {
  color: #666;
}

h2 {
  text-align: left;
  font-weight: bold;
  padding: 5px 5px 5px 15px;
  font-size: 12px;
  margin: -7px;
  margin-bottom: 0px;
  background-color: #c00;
  color: #fff;
}

ul li {
  font-size: 12px;
  list-style: square;
}
/*
 * This is a manifest file that'll be compiled into application.css, which will include all the files listed below.
 *
 * Any CSS and SCSS file within this directory, lib/assets/stylesheets, vendor/assets/stylesheets,
 * or any plugin's vendor/assets/stylesheets directory can be referenced here using a relative path.
 *
 * You're free to add application-wide styles to this file and they'll appear at the bottom of the
 * compiled file so the styles you add here take precedence over styles defined in any
 * defined in the other CSS/SCSS files in this directory. It is generally better to create a new file per style scope.
 *
 * require_tree .
 * require_self
 */
**BASIC MODELS**

---

File D:\railsApps\playgroundJan2016\app\models\answer.rb
class Answer < ActiveRecord::Base
  belongs_to :enroll
  has_many :kts_covered_by_answers
end

5

File D:\railsApps\playgroundJan2016\app\models\enroll.rb
class Enroll < ActiveRecord::Base
  belongs_to :Student
  has_many :answers
  has_many :prerequisites
  has_many :program_of_studies
end

7

File D:\railsApps\playgroundJan2016\app\models\student.rb
class Student < ActiveRecord::Base
  has_many :enrolls
  has_many :answers, through: :enrolls
end

5

File D:\railsApps\playgroundJan2016\app\models\prerequisite.rb
class Prerequisite < ActiveRecord::Base
  belongs_to :Enroll
end

5

File D:\railsApps\playgroundJan2016\app\models\class_average.rb
class ClassAverage < ActiveRecord::Base
end

3

File D:\railsApps\playgroundJan2016\app\models\knowledge_topic.rb
class KnowledgeTopic < ActiveRecord::Base
  has_many :slo_covered_by_kts
  has_many :student_learning_outcomes, :through => :slo_covered_by_kts
  has_many :answers, :through => :kts_covered_by_answers
  has_many :kts_covered_by_answers
end

7

File D:\railsApps\playgroundJan2016\app\models\program_of_study.rb
class ProgramOfStudy < ActiveRecord::Base
  belongs_to :Enroll
end

4

File D:\railsApps\playgroundJan2016\app\models\slo_covered_by_kt.rb
class SloCoveredByKt < ActiveRecord::Base
  belongs_to :student_learning_outcome
  belongs_to :knowledge_topic
end

5
class PeoCoveredBySlo < ActiveRecord::Base
  belongs_to :StudentLearningOutcome
  belongs_to :ProgramEducationalObjective
end

class KtsCoveredByAnswer < ActiveRecord::Base
  belongs_to :knowledge_topic
  belongs_to :answer
end

class StudentLearningOutcome < ActiveRecord::Base
  has_many :slo_covered_by_kts,
    dependent => :destroy
  has_many :knowledge_topics,
    through: :slo_covered_by_kts
  has_many :peo_covered_by_slos
  has_many :program_educational_objectives,
    through: :peo_covered_by_slos
end

class ProgramEducationalObjective < ActiveRecord::Base
  has_many :peo_covered_by_slos
  has_many :student_learning_outcomes,
    through: :peo_covered_by_slos
end
def self.slo_coverage_for_cs()
  courses_and_kt = Enroll.joins(:answers).select('enrolls.course, answers.knowledge_topic_id').uniq
  results = match_course_to_slo(courses_and_kt)
  results = results_reorder(results)
  return results
end

def self.slo_coverage_for_ceg()
  courses_and_kt = Enroll.joins(:answers).select('enrolls.course, answers.knowledge_topic_id').uniq
  results = match_course_to_slo(courses_and_kt)
  results = results_reorder(results)
  return results
end
End Quiz Coverage Report
Developed

############################
56
57
58
############################
Start Summary Report Developed

############################
59
def self.get_semester()
  date = Date.today
  month = date.mon
  if month < 5
    return 'Spring'
  end
  if month < 8
    return 'Summer'
  end
  return 'Fall'
end
71
def self.get_year_offered()
  return Date.today.year
end
75
def self.get_current_course_average(current_semester, current_year)
  current_averages = Array.new
  courses = get_unique_compact_course_list
  courses.each { |course|
    current = ClassAverage.find_by(semester: current_semester, year_offered: current_year, course: course)
    if current == nil
      current_averages.push 105
    else
      current_averages.push current.quiz_average
    end
  }
  return current_averages
end
90
def self.get_unique_compact_course_list()
  courses = ClassAverage.all.order(:course).pluck(:course).uniq!
  return courses
end
97
def self.get_course_boxplot_data(current_semester, current_year)
  final_table = Array.new
  boxplot_table = Array.new
  past_data_table = Array.new
  courses = get_unique_compact_course_list
  boxplot_table.push(courses)
  course_index = 0
  courses.each { |course|
    course_sections = ClassAverage.where(course: course)
    quiz_scores = course_sections.pluck(:quiz_average)
    current = ClassAverage.find_by(semester: current_semester, year_offered: current_year, course: course)
    unless current == nil
      current_index = quiz_scores.index(current.quiz_average)
      unless current_index == nil
        quiz_scores.delete_at(current_index)
      end
    end
  }
  return final_table
end
106
course.each { |course|
unless quiz_scores.empty?
require 'descriptive_statistics'
past_data_table = scores_to_points(course_index, quiz_scores, past_data_table)
course_index = course_index + 1
boxplot_points = [quiz_scores.percentile(0), quiz_scores.percentile(25), quiz_scores.percentile(50), quiz_scores.percentile(75), quiz_scores.percentile(100)]
boxplot_table.push boxplot_points
125
126  end
127
} final_table.push boxplot_table
final_table.push
past_data_table
return final_table
end
133

def self.get_past_quiz_scores(course, current_semester, current_year)
quiz_scores = ClassAverage.where(course: course).pluck(:quiz_average)
current = ClassAverage.find_by(semester: current_semester, year_offered: current_year, course: course)
137  current_index = quiz_scores.index(current.quiz_average)
unless current_index.equal? nil
quiz_scores.delete_at(current_index)
end
return quiz_scores
end
143
144  def self.scores_to_points(course_index, quiz_scores, past_data_table)
145
quiz_scores.each { |score|
point = [course_index, score]
past_data_table.push point
}
return past_data_table
end
152

def self.get_slo_averages(semester, year_offered)
fill_in_slo_averages_table(semester, year_offered)
results = SloAverage.where.not(title: 'n').where(semester: semester, year_offered: year_offered)
return results
end
159

def self.fill_in_slo_averages_table(semester, current_year)
search_year = current_year
semesters = ['Spring', 'Summer', 'Fall']
index = semesters.index(semester)
results = nil
165
#find the last term entered
while ((results == nil) || (results < 1)) do
results = SloAverage.where.not(title: 'n').where(semester: semesters[index], year_offered: search_year).count
index = index - 1
if index < 0
index = 2
search_year = search_year - 1 end
end

index = index + 1
if index > 2
index = 0
search_year = search_year + 1 end

#Adding CEG Data next 2 lines added
index = 1
search_year = 2012

until ((semester.include?(semesters[index]) &&
(search_year == current_year))
do
index = index + 1
if index > 2
index = 0
search_year = search_year + 1 end

add_data_in_slo_averages_table(semesters[index], search_year)

return end

def self.add_data_in_slo_averages_table(semester, year_to_add)
all_answers_for_term = Enroll.where(semester: semester,
year_to_add: year_to_add)
connect_answers_to_slos = all_answers_for_term.joins(answers: {kts_covered_by_answers:
(knowledge_topic: {slo_covered_by_kts:
:student_learning_outcome}}).select('enrolls.*, answers.*,
knowledge_topics.*, kts_covered_by_answers.*,
student_learning_outcomes.*')
temp_result = connect_answers_to_slos.select(:correct,
:accredidation_body, :title, 'count(*) AS total_answers').group(:correct,
:accredidation_body, :title)
temp_result = temp_result.order(correct: :asc)
results = Array.new
206 temp_result.each { |row|
if row.correct == 0
results.push([row.accredidation_body, row.title, 0,
row.total_answers])
else
found = true
results.each { |solution|
if solution[0] == row.accredidation_body and
solution[1] == row.title
solution[2] = row.total_answers
found = false
end
end
if !found
results.push([row.accredidation_body, row.title,
row.total_answers, row.total_answers])
end
end

results.each { |row|
score = ((row[2].to_f / row[3]) * 100)
item = SloAverage.find_or_create_by(title: row[1], program: row[0], semester: semester, year_offered: year_to_add)
item.average = score
item.number_students = row[3]
item.save

return results

def self.get_slo_table(current_semester, current_year)
final_table = Array.new
boxplot_table = Array.new
past_data_table = Array.new
slos = SloAverage.where.not(title: 'n').select(:program, :title).order(:program, :title).distinct
titles_and_programs = Array.new
slos.each { |row|
titles_and_programs.push(row.program + "": " + row.title)
}
boxplot_table.push(titles_and_programs)
slo_index = 0
slos.each { |slo|
slo_scores = SloAverage.where(program: slo.program, title: slo.title).pluck(:average)
current = SloAverage.find_by(semester: current_semester, year_offered: current_year, program: slo.program, title: slo.title)
if current
  current_index = slo_scores.index(current.average)
  unless current_index == nil
    slo_scores.delete_at(current_index)
  end
unless slo_scores.empty?
  require 'descriptive_statistics'
  past_data_table = scores_to_points(slo_index, slo_scores, past_data_table)
  slo_index = slo_index + 1
  boxplot_points = [slo_scores.percentile(0), slo_scores.percentile(25), slo_scores.percentile(50), slo_scores.percentile(75), slo_scores.percentile(100)]
  boxplot_table.push(boxplot_points)
  slo_index = slo_index + 1
end
end
final_table.push(boxplot_table)
final_table.push(past_data_table)
return final_table

end

def self.create_course_summary_table(current_averages, current_year, current_semester)
  table = Array.new
current_averages.each { |row|
  require 'descriptive_statistics'
  data_points = get_past_quiz_scores(row.course, current_semester, current_year)
  mean = data_points.mean
  std_dev = data_points.standard_deviation
  significant = false
  difference = (row.quiz_average - mean).abs
  if difference > (2*std_dev)
    significant = true
  end
  new_row = [(row.program + row.course.to_s), row.number_students, row.quiz_average.round(2), data_points.number.to_i, mean.round(2), std_dev.round(2), significant]
  table.push new_row
} return table
end

def self.create_slo_summary_table(current_averages, current_year, current_semester)
  table = Array.new
  current_averages.each { |row|
    require 'descriptive_statistics'
    data_points = get_past_slo_scores(row.program, row.title, current_semester, current_year)
    mean = data_points.mean
    std_dev = data_points.standard_deviation
    significant = false
    difference = (row.average - mean).abs
    if difference > (2*std_dev)
      significant = true
    end
    new_row = [(row.program + row.title), row.number_students, row.average.round(2), data_points.number.to_i, mean.round(2), std_dev.round(2), significant]
    table.push new_row
  } return table
end

def self.get_past_slo_scores(program, title, current_semester, current_year)
  quiz_scores = SloAverage.where.not(title: 'n').where(program: program, title: title).pluck(:average)
  current = SloAverage.where.not(title: 'n').find_by(semester: current_semester, year_offered: current_year, program: program, title: title)
  current_index = quiz_scores.index(current.average) unless current_index == nil
  quiz_scores.delete_at(current_index)
  return quiz_scores
end

def self.get_course_breakdown(averages)
  significant = Array.new
  averages.each { |row|
    if row[6]
      significant.push row[0]
    end
  } return significant
end

def self.get_course_breakdown(courses_to_breakdown, semester, current_year)
  charting_data = Array.new
  courses_to_breakdown.each { |row|
    if row[6]
      significant.push row[0]
    end
  } return significant
end
courses_to_breakdown.each { |course|
  first_number = 2
  if course[2] =~ /[[:alpha:]]/  
    first_number = 3
  end
  program = course.slice(0, first_number)
  course_num = course.slice(first_number, (course.length - first_number))
  course_data = get_course_breakdown_data(program, course_num, semester, current_year)
  charting_data.push course_data }
return charting_data

349
end

def self.get_course_breakdown_data(program, course_num, semester, current_year)
  all_students_for_course = Enroll.where(course: (program + ' ' + course_num.to_s), semester: semester, year_offered: current_year)
  answers = all_students_for_course.joins(:answers).select('enrolls.*, answers.*')
  temp_result = answers.select(:correct, :question_text, 'count(*) AS total_answers').group(:correct, :question_text)
  results = Array.new
  temp_result.each { |row|
    if row.correct == 0
      results.push([row.question_text, 0, row.total_answers])
    else
      found = false
      results.each { |solution|
        if solution[0] == row.question_text
          solution[1] = row.total_answers
          found = true
          break
        end
      } unless found
      results.push([row.question_text, row.total_answers, row.total_answers])
    end
  end
end
return results end

def self.get_slo_breakdown(slos_to_breakdown, semester, current_year)
    charting_data = Array.new
    slos_to_breakdown.each
      |title|
      program = title.split('[:').first
      slo = title.split(' ').last
      slo_data = get_slo_data(program, slo, semester, current_year)
      charting_data.push slo_data
    return charting_data
end

def self.get_slo_data(program, slo, semester, current_year)
    all_answers_for_term = Enroll.where(semester: semester, year_offered: current_year)
    connect_answers_to_slos = all_answers_for_term.joins(answers: {knowledge_topic: {slo_covered_by_kts: :student_learning_outcome}}).select('enrolls.*, answers.*, knowledge_topics.*, student_learning_outcomes.*')
    answers_to_slo = connect_answers_to_slos.where("student_learning_outcomes.accreditation_body = ? AND student_learning_outcomes.title = ?", program, slo)
    area_results = add_knowledge_areas(answers_to_slo)
    unit_results = add_knowledge_units(answers_to_slo)
    topic_results = add_knowledge_topics(answers_to_slo)
    question_results = add_knowledge_topics_questions(answers_to_slo, topic_results)
    results = [area_results, unit_results, topic_results, question_results]
    return results
end

def self.add_knowledge_areas(answers_to_slo)
    results = Array.new
    temp_result = answers_to_slo.select(:correct, :knowledge_area, 'count(*) AS total_answers').group(:correct, :knowledge_area)
    temp_result.each
      |row|
      if row.correct == 0
        results.push([row.knowledge_area, 0, row.total_answers])
      else
        found = false
        results.each
          |solution|
          if solution[0] == row.knowledge_area
            solution[1] = row.total_answers
            found = true
          end
        unless found
          results.push([row.knowledge_area, row.total_answers, row.total_answers])
        end
      end
    end
    return results
end

def self.add_knowledge_units(answers_to_slo)
    results = Array.new
    temp_result = answers_to_slo.select(:correct, :knowledge_unit, 'count(*) AS total_answers').group(:correct, :knowledge_unit)
    temp_result.each
      |row|
      if row.correct == 0
        results.push([row.knowledge_unit, 0, row.total_answers])
      else
        found = false
        results.each
          |solution|
          if solution[0] == row.knowledge_unit
            solution[1] = row.total_answers
            found = true
          end
        unless found
          results.push([row.knowledge_unit, row.total_answers, row.total_answers])
        end
      end
    end
    return results
end

question_results = add_knowledge_topics_questions(answers_to_slo, topic_results)
results = [area_results, unit_results, topic_results, question_results]
return results
end

def self.add_knowledge_units(answers_to_slo)
results = Array.new
temp_result = answers_to_slo.select(:correct, :knowledge_area, :knowledge_unit, 'count(*) AS total_answers').group(:correct, :knowledge_area, :knowledge_unit)
temp_result.each { |row|
  if row.correct == 0
    results.push([row.knowledge_area, row.knowledge_unit, 0, row.total_answers])
  else
    found = false
    results.each { |solution|
      if solution[0] == row.knowledge_area and solution[1] == row.knowledge_unit
        solution[2] = row.total_answers
        found = true
      end
    }
    unless found
      results.push([row.knowledge_area, row.knowledge_unit, row.total_answers, row.total_answers])
    end
  end
}
return results
end

def self.add_knowledge_topics(answers_to_slo)
  results = Array.new
topic_results.each { |row|
  if ((row[3].to_f / row[4] * 100).round(1)) < 60
    this_topic = [row[0], row[1], row[2], row[3], row[4]]
    results.push([this_topic[0], this_topic[1], 0, this_topic[3], this_topic[4]])
  else
    found = false
    results.each { |solution|
        found = true
      end
    }
    unless found
      results.push([row[0], row[1], row[2], row[3], row[4]])
    end
  end
}
return results
end
row[2]) topics.push
this_topic end
}
501
502
503 topics.each { |knowledge_topic_row| 504 answers_to_topic = answers_to_slo.where("knowledge_topics.knowledge_area = ? and knowledge_unit = ? and knowledge_topics.knowledge_topic = ?", knowledge_topic_row[0], knowledge_topic_row[1], knowledge_topic_row[2]) temp_result = answers_to_topic.select(:course, :question_text, :correct, :knowledge_area, :knowledge_unit, :knowledge_topic, 'count(*) AS total_answers').group(:course, :question_text, :correct, :knowledge_area, :knowledge_unit, :knowledge_topic) 507 temp_result.each { |row| if row.correct == 0 results.push([row.course, row.question_text, row.knowledge_area, row.knowledge_unit, row.knowledge_topic, row.total_answers]) else found = false
results.each { |solution|
529
530

End Summary Report Developed

## Begin Single Course Report

```ruby
def self.get_pedagogy_comparison(program, course_num, old_terms, new_terms)
all_students_for_course = Enroll.where(course: (program + ' ' + course_num.to_s), semester: 'spring', year_offered: 2015)
return all_students_for_course
end
```

```ruby

solution[4] == row.knowledge_topic
solution[5] = row.total_answers
found = true end
}
unless found
results.push([row.course, row.question_text, row.knowledge_area, row.knowledge_unit, row.knowledge_topic, row.total_answers])
end end
}
}
526
return results end
529
530
End Summary Report Developed

## Begin Single Course Report

```ruby
def self.get_pedagogy_comparison(program, course_num, old_terms, new_terms)
```
### 538

def self.get_course_data(program, course_num, students):
    all_students_for_course = Enroll.where(course: (program + ' ' + course_num.to_s), student_id: students).uniq
    results = get_course_data_for_students(all_students_for_course)
    return results

543

544 end

def self.get_course_data_for_students(all_students_for_course):
    answers = all_students_for_course.joins(:answers).select('enrolls.*, answers.*').uniq
    temp_result = answers.select(:correct, :question_text, 'count(*) AS total_answers').group(:correct, :question_text)

549
results = Array.new
    temp_result.each { |row|
    if row.correct == 0
        results.push([row.question_text, 0, row.total_answers])
    else
        found = false
        results.each { |solution|
            if solution[0] == row.question_text
                solution[1] = row.total_answers
                found = true
            end
        end
        unless found
            results.push([row.question_text, row.total_answers, row.total_answers])
        end
    end
end

568
return results

571 def self.get_student_list(program, course_num, semester, current_year, faculty_last_name):
    students = Enroll.where(course: (program + ' ' + course_num.to_s), semester: semester, year_offered: current_year, faculty_last_name: faculty_last_name).pluck(:student_id)
    enrolls_ids = Prerequisite.where(course: (program + ' ' + course_num.to_s), semester: semester, year_taken: current_year, faculty_last_name: faculty_last_name).pluck(:enroll_id)
    students_list_2 = Enroll.where(id: enrolls_ids).pluck(:student_id)
    students = students.concat(students_list_2).uniq
    return students

580 def self.get_single_course_chart_data(table):

272
title = Array.new
correct = Array.new
incorrect = Array.new
table.each { |row|
title.push ( row[0][0, 15]+'...')
correct.push row[1]
incorrect.push (row[2] - row[1])
}
results = Array.new
results.push title
results.push correct
results.push incorrect
return results
end

### End Single Course Report

#### BEGIN

def self.courses_knowledge_topics_summary_results(terms)
    answers_to_slo = Enroll.joins(answers:
    {kts_covered_by_answers:
    {knowledge_topic:
    {slo_covered_by_kts: :student_learning_outcome}}}).se
    results = Array.new
    temp_result = answers_to_slo.select(:course,
    :correct, :semester,
    :year_offered,
    'knowledge_topics.
    knowledge_area,
    knowledge_topics.knowledge_topic,
    student_learning_outcomes.title
    ).count(*) AS total_answers').group(:course,
    :correct, :semester,
    :year_offered, :knowledge_area,
    :knowledge_unit,
    :knowledge_topic,
    :accreditation_body,
    :title).order(:course,
    'knowledge_topics.knowledge_area',
    'knowledge_topics.knowledge_unit', 'knowledge_topics.
    knowledge_topic',
    'student_learning_outcomes.accreditation_body',
    'student_learning_outcomes.title', 'answers.
    correct', 'total_answers',
    :semester, :year_offered)
    temp_result.each { |row|
        slo = row.accreditation_body.concat(':
        :'.concat(row.title))
        term = row.semester.concat(':
        :'.concat(row.year_offered.to_s))
    }
elsif row.semester
term = row.semester
elsif row.year_offered
term = row.year_offered
else
term = ''
end

if row.correct == 0
results.push([row.course, row.knowledge_area, row.knowledge_unit, row.knowledge_topic, 0, row.total_answers, slo, term])
else
found = false
results.each { |solution|
solution[4] = row.total_answers
found = true
end
}
end

results.push([row.course, row.knowledge_area, row.knowledge_unit, row.knowledge_topic, row.total_answers, row.total_answers, slo, term])
end
end

results = combine_results_lines(results, terms)
return results

def self.combine_results_lines(temp_results, terms)
    temp_results = clean_slos(temp_results)
    results = clean_terms(temp_results, terms)
    return results
end

def self.clean_slos(temp_results)
    results = Array.new
    temp_results.each { |row|
found = false
results.each { |new_solution|
found = true
end
}
end

results.push([row.course, row.knowledge_area, row.knowledge_unit, row.knowledge_topic, row.total_answers, row.total_answers, slo, term])
end
end

644 results = combine_results_lines(results, terms)
645 return results
end
648 def self.combine_results_lines(temp_results, terms)
650 temp_results = clean_slos(temp_results)
652 results = clean_terms(temp_results, terms)
654 return results
658 end
659 def self.clean_slos(temp_results)
results = Array.new
temp_results.each { |row|
found = false
results.each { |new_solution|
found = true
end
}
end

results.push(row)
end

675
results.each { |row|
  list = row[6].split(',
')
  cac_list = ''
  eac_list = ''
  list.each { |entry|
    if entry.split(':') [0] == 'CAC'
      cac_list = cac_list.concat(entry.split(':
')) [1]
    else
      eac_list = eac_list.concat(entry.split(':
')) [1]
    end
  }
  if cac_list.length > 0 and eac_list.length > 0
    row[6] = 'CAC: ' + cac_list + ',
    EAC: ' + eac_list
  elsif cac_list.length > 0
    row[6] = 'CAC: ' + cac_list
  else
    row[6] = 'EAC: ' + eac_list
  end
}

return results
end

self.clean_terms(temp_results, terms)
results = Array.new

temp_results.each { |row|
  found = false
  term_index = terms.index row[7]
  results.each { |new_solution|
    if new_solution[0] == row[0] and
      new_solution[1] == row[1] and
      new_solution[2] == row[2] and
      new_solution[term_index+5] = [row[5], ((row[4].to_f/row[5] * 100).round(1))]
      found = true
    end
  }
  unless found
    new_row = row[0..3].push(row[6]).concat(Array.new(15, Array.new(2,0)))
    new_row[term_index+5] = [row[5], ((row[4].to_f/row[5] * 100).round(1))]
    results.push(new_row)
  end
}

return results
end

def self.get_array_of_terms_for_last_five_years()
  terms = Array.new
  year_index = get_year_offered
  current_term = get_semester
  if current_term == 'Spring'
    semester_index = 0
  elsif current_term == 'Summer'
    semester_index = 1
  else
    semester_index = 2
  end
  semesters = ['Spring ', 'Summer ', 'Fall ']
  number_terms_needed = 15
  while (number_terms_needed > 0)
    do
      terms.unshift(semesters[semester_index] + (year_index.to_s))
      semester_index = semester_index - 1
  end
end
if(semester_index < 0)
semester_index = 2
year_index = year_index - 1
end
number_terms_needed = number_terms_needed - 1
end
return terms
end

##############################
#### END
abet_data_collection_summary_c
ourse_focus

##############################################################################
###  END
756
757 758 759 760
761
762 end
763
class LoadInternalReport
  def self.importing_ir_report(file)
    sheet1 = open_spreadsheet(file)
    header_row_index = get_header_row_index(sheet1)
    course_header_row = sheet1.row(header_row_index)
    course_list = create_course_list(course_header_row)
    campus_id_index = course_header_row.index 'Campus Id'
    index = header_row_index + 1
    #Skip row of headers
    number_of_lines = sheet1.last_row
    starting_course_column = course_list[0][0]
    12
    # loop for each student in the file
    while index < number_of_lines do
      row = sheet1.row(index)
      stu = Student.find_or_create_by(w_number: row[campus_id_index])
      column = starting_course_column
      length_course_cell = row.length - 5
      while column < length_course_cell do
        cell_content = row[column]
        if cell_content
          results = divide_cell_content(cell_content)
          semester = results[0]
          year_taken = results[1]
          grade = results[2]
          if grade
            course = find_course_association(column, course_list)
            if course.eql? 'CS 1160'
              stu.update(path_1160: true)
            end
            load_student_grade_data(stu, semester, year_taken, grade, course)
          end
          column += 1
        end
      end
      stu.act = row[length_course_cell + 4]
      stu.act_math = row[length_course_cell + 3]
      stu.mpl = row[length_course_cell + 2]
      stu.high_school_gpa = row[length_course_cell]
      stu.save
      index = index + 1
    end
  end

  def self.get_header_row_index(sheet)
    i = 0
    size = sheet.last_row
    while i < size do
      row = sheet.row(i)
      if row[0] and row[0].include? 'ID'
        return i
      end
      i += 1
    end
    return -1
  end

  def self.create_course_list(row)
    # code for creating course list
  end
end

MODEL: LOAD INTERNAL REPORT

# code for creating course list

course_list = Array.new
i = 0
size = row.length
while i < size do
  if row[i]
    program = get_program_from_string(row[i])
    number = get_number_from_string(row[i])
    unless program.include? 'NONE'
      course_list.push [i, program, number]
    end
  end
  i += 1
end
return course_list
end

def self.open_spreadsheet(file)
case File.extname(file.original_filename)
when '.csv' then
  Roo::CSV.new(file.path)
when '.xls' then
  Roo::Excel.new(file.path)
when '.xlsx' then
  Roo::Excelx.new(file.path)
else
  raise "Unknown file type: #{file.original_filename}"
end
end

def self.divide_cell_content(cell_content)
tmp = cell_content.split('/')
tmp2 = tmp[0].split(' ') 97
semester = tmp2[0]
year_taken = tmp2[1]
end

def self.get_program_from_string(course)
  (tmp.length > 1) ? grade = tmp[1] : grade = nil
  unless grade.eql? 'A' or grade.eql? 'a' or grade.eql? 'B'
    or grade.eql? 'b' or grade.eql? 'C' or grade.eql? 'c'
    or grade.eql? 'D' or grade.eql? 'd'
    or grade.eql? 'E' or grade.eql? 'f'
    or grade.eql? 'W' or grade.eql? 'W'
  end
  grade = nil
results = [semester, year_taken, grade]
return results
end

def self.find_course_association(column, course_list)
course_index = 0
course_value = course_list[course_index][0]
while column >= course_value
  course_index += 1
  break unless course_index < course_list.length
  course_value = course_list[course_index][0]
end

course_index -= 1

course = course_list[course_index][1] + " " +
(course_list[course_index][2]).to_s
return course
end

def self.get_program_from_string(course)
program = 'NONE'

if course.include? 'CS'
  program = 'CS'
elsif course.include? 'CEG'
  program = 'CEG'
elsif course.include? 'MTH'
  program = 'MTH'
elsif course.include? 'EGR'
  program = 'EGR'
end

return program

def self.get_number_from_string(course)
  number = 9999
  if course.include? 'CS'
    parts = course.partition('CS')
    number = parts[2].strip.to_i
  elsif course.include? 'CEG'
    parts = course.partition('CEG')
    number = parts[2].strip.to_i
  elsif course.include? 'MTH'
    parts = course.partition('MTH')
    number = parts[2].strip.to_i
  elsif course.include? 'EGR'
    parts = course.partition('EGR')
    number = parts[2].strip.to_i
  end

  return number
end

def self.load_student_grade_data(student, semester, year_taken, grade, course)
  enrolled = Enroll.find_or_initialize_by(course: course, student_id: student.id, semester: semester, year_offered: year_taken)
  enrolled.grade = grade
  enrolled.save
end

279
```ruby
class LoadSloCoveredByKt
  def self.import_slo_covered_by_kt(file)
    sheet1 = open_spreadsheet(file)
    index = 2 # Skip row of headers
    numberOfLines = sheet1.last_row
    # loop for each student in the file
    while (index < numberOfLines) do
      row = sheet1.row(index)
      slo = StudentLearningOutcome.find_or_create_by(accredidation_body: row[3], title: row[4])
      kt = get_knowledge_topic(row)
      newsloktrow = SloCoveredByKt.find_or_create_by(knowledge_topic_id: kt.id, student_learning_outcome_id: slo.id)
      newsloktrow.knowledge_topic_id = kt.id
      newsloktrow.student_learning_outcome_id = slo.id
      newsloktrow.save
      index = index + 1
    end # end loop while (currentRowIndex < numberOfLines) do
  end
end
```
def self.get_knowledge_topic(row)
    kt_excel = row[2].gsub(/\([;:\]/, '').gsub(/\(.*\)/, '').strip
    kt_excel = kt_excel[0, 250]
    kt_excel = row[2][0, 250]
    #kt_excel = row[2][0, 250]

    KnowledgeTopic.where(knowledge_area: row[0]).order(:id).find_each do |item|
        kt = item.knowledge_topic
        if kt.include?(kt_excel)
            return item
        end
    end

    KnowledgeTopic.create(knowledge_area: row[0], knowledge_unit: row[1], knowledge_topic: kt_excel)
end

class LoadStudentQuizResult
  def self.start_quiz_upload(file, crn, course, course_number, instr_last_name, instr_first_name, semester, course_year, section)
    crn = crn.to_s + course_year.to_s
    sheet1 = open_spreadsheet(file)
    current_row_index = 2  # Skip row of headers
    number_of_lines = sheet1.last_row
    # loop for each student in the file
    while current_row_index < number_of_lines do
      results = clean_and_pull_student_data(sheet1, current_row_index, number_of_lines)
      current_row_index = results[1]
      student_data_array = results[0]
      handle_a_student(student_data_array, crn, instr_last_name, instr_first_name, course, course_year, semester, section)
    end  # end student loop
    while (current_row_index < number_of_lines) do
      end
    self.open_spreadsheet(file)
    case File.extname(file.original_filename)
    when '.csv' then
      Roo::CSV.new(file.path)
    when '.xls' then
      Roo::Excel.new(file.path)
    when '.xlsx' then
      Roo::Excelx.new(file.path)
    else
      raise "Unknown file type: #{file.original_filename}"
    end
    def self.handle_a_student(student_data_array, crn, instr_last_name, instr_first_name, course, course_year, semester, section)
      crn = crn.to_s + course_year.to_s
    end
end
end
return if previously_taken(student_data_array)

student_index = 0

current_row = student_data_array[student_index]

uid = current_row[0]
total_number_questions = 0
number_questions_correct = 0

student = Student.find_or_create_by(w_number: uid)
enrolled = Enroll.find_or_create_by(student_id: student.id, course: course, semester: semester, year_offered: course_year)

enrolled.update(course_registration_number: crn, section: section, faculty_first_name: instr_first_name, faculty_last_name: instr_last_name)

while student_index < student_data_array.length do
  current_row = student_data_array[student_index]

  if current_row[10].eql? 'Demographics'
    student_index = demographics_info(student_data_array, (student_index + 1), student, enrolled)
  elsif (answer_selected)
    student_data.push current_row
  end

  current_row_index += 1
  current_row = sheet1.row(current_row_index)

  while current_row.empty? && (current_row[0].eql? uid) do
    section_header = !(current_row[7].eql? nil)
    answer_selected = !(current_row[15].eql? 'UnChecked')

    if (section_header)
      inside_section = !inside_section
      temp_array = handle_quiz_questions(student_data_array, (student_index + 1), total_number_questions, number_questions_correct, enrolled, student)
    end

    student_index = temp_array[0]
total_number_questions = temp_array[1]
number_questions_correct = temp_array[2]

end #end student technical sections loop

while (current_row[0].eql? uid) do
  # END LOAD STUDENT QUIZ RESULTS
  load_enroll(enrolled, total_number_questions, number_questions_correct)

end

#makes an array of one students data rows. The only rows included are section dividers and checked material

def self.clean_and_pull_student_data(sheet1, current_row_index, max_lines)
  current_row = sheet1.row(current_row_index)

  if (section_header)
    inside_section = !inside_section
    temp_array = handle_quiz_questions(student_data_array, (student_index + 1), total_number_questions, number_questions_correct, enrolled, student)
  end

  current_row_index += 1
  current_row = sheet1.row(current_row_index)

  while current_row.empty? && (current_row[0].eql? uid) do
  end
do 81  current_row_index += 1
    current_row = sheet1.row(current_row_index)
end
84
85  end
86
if inside_section
    student_data = cleanup_to_remove_partial_final_section(student_data)
end
90
91  return [student_data, current_row_index] 92  end

93
def self.cleanup_to_remove_partial_final_section(student_data)
    inside_section = false
    current_row_index = 0
    last_summary_index = -1
    length = student_data.length
    while current_row_index < length
        do
            current_row = student_data[current_row_index]
            section_header = !(current_row[7].eql? nil)
            if (section_header &&
                inside_section)
                inside_section = !inside_section
            end
            last_summary_index = current_row_index
            elsif section_header
                inside_section = !inside_section
            end
            current_row_index += 1
end

112
student_data = student_data.first(last_summary_index + 1)
return student_data
end
116
117
def self.previously_taken(student_data)
    student_data.each { |row_content|
        if row_content[10].eql? 'Initial experience'
            if row_content[14].include? 'No'
                return true
            end
        else
            return false
        end
    } return true
129
130  end
131
#returns the first line of a new section
def self.handle_quiz_questions(student_data, current_row_index, total_num_questions, num_quest_correct, enrolled, student)
    current_row = student_data[current_row_index]
    until current_row[7]
        if current_row[10].include? 'Prerequisite course'
            current_row_index = handle_prerequisite_classes(student_data, current_row_index, enrolled, student)
        end
    end
    current_row_index += 1
end
```ruby
current_row = student_data[current_row_index]
elsif current_row[10]
  temp_array = handle_technical_questions(student_data, current_row_index, total_num_questions, num_quest_correct, enrolled)
  current_row_index = temp_array[0]
  total_num_questions = temp_array[1]
  num_quest_correct = temp_array[2]
  current_row = student_data[current_row_index]
end

current_row_index = current_row_index + 1
return [current_row_index, total_num_questions, num_quest_correct]
end

#returns the section summary

def self.handle_technical_questions(student_data, current_row_index, total_num_questions, num_quest_correct, enrolled)
  current_row = student_data[current_row_index]
  #Each Question in this section
  until current_row[7]
    question_title = current_row[11]
    question_kts = load_knowledge_topic(current_row)
    question_answer = clean_question_answer(current_row[14])
    if current_row[16].to_i > 0
      num_quest_correct += 1
    else
      correct = false
    end
    169
    load_answer(question_title, question_answer, enrolled, correct, question_kts) 171
    current_row_index += 1
    current_row = student_data[current_row_index]
  end
  total_num_questions += 1
end

return [current_row_index, total_num_questions, num_quest_correct]
end

def self.handle_prerequisite_courses(student_data, current_row_index, enrolled, student)
  current_row = student_data[current_row_index]
  pre_course = current_row[14]
  current_row_index += 1
  current_row = student_data[current_row_index]
  pre_year = current_row[15]
  current_row_index += 1
  current_row = student_data[current_row_index]
  pre_semester = 'Other'
  if current_row[10].include? 'Term of prerequisite course'
    pre_semester = current_row[14]
  end
  current_row_index += 1
end
```
current_row = student_data[current_row_index] end
pre_instructor = current_row[15]
current_row_index += 1
load_prerequisite(enrolled, pre_course, pre_year, pre_semester, pre_instructor, student)
return current_row_index end
#returns row index for first row of next section
def self.demographics_info(student_data, current_row_index, student, enrolled)
    row_content = student_data[current_row_index]
    programs_of_study = Array.new
    first_name = row_content[2]
    last_name = row_content[3]
    academic_progress = 'Other, including non-degree seeking students'
gender = 'I prefer not to provide this information.'
ethnicity = 'I prefer not to provide this information'
    until row_content[10].eql? 'Summary: Demographics'
        if row_content[10].include? 'Program of study'
            programs_of_study.push row_content[14]
        elsif row_content[10].include? 'Academic progress'
            academic_progress = row_content[14]
        end
    end
    end
end
load_program_of_study(programs_of_study, enrolled)
current_row_index += 1
return current_row_index
end
def self.load_enroll(enrolled, total_num_questions, num_quest_correct)
score = (num_quest_correct.to_f/total_num_questions) * 100
enrolled.quiz_score = score
enrolled.save

course = enrolled.course.split(' ')
class_avg = ClassAverage.find_by(program: course[0], course: course[1], semester: enrolled.semester,
year_offered: enrolled.year_offered)
unless class_avg
    class_avg = ClassAverage.create(program: course[0], course: course[1],
        semester: enrolled.semester, year_offered: enrolled.year_offered,
        number_students: 0, quiz_average: 0)
end

class_avg.quiz_average = ((class_avg.number_students*class_avg.quiz_average) + score)/(class_avg.
    number_students + 1)
class_avg.number_students = class_avg.number_students + 1
class_avg.save

end

def self.load_section(crn, instr_last_name, instr_first_name, course,
    course_number, course_year, semester, section_number)
    s = Section.find_or_create_by(course_registration_number: crn)
    f = Faculty.find_or_create_by(last_name: instr_last_name, first_name: instr_first_name)
    c = Course.find_or_create_by(program: course, number: course_number)
    s.Faculty = f
    s.Course = c
    s.year_offered = course_year
    s.semester = semester
    s.section = section_number
    s.save
    return s
end

end

def self.load_prerequisite(enrolled, pre_course, pre_year, pre_semester, pre_instructor,
    student)
    pre_course = clean_course_string(pre_course)
    pre_program = clean_program_from_string(pre_course)
    pre_number = clean_number_from_string(pre_course)
    c = pre_program + " " + pre_number.to_s
    if c.include? 'CS 1160'
        student.path_1160 = true
        student.save
    end
    p = Prerequisite.find_or_create_by(enroll_id: enrolled.id, course: c)
    p.year_taken = pre_year
    p.semester = pre_semester
    pre_instructor = clean_instructor(pre_instructor)
    p.faculty_last_name = pre_instructor
    p.save
end

def self.load_knowledge_topic(row)
    whole_row = row.join(',')
    #get through the date slashes and in doing so, remove names
    i = 0

while i < 4
slash_index = whole_row.index('/')
whole_row = whole_row[slash_index + 1, whole_row.length - slash_index]
i += 1
end
mc_index = whole_row.index('MC')
unless mc_index
mc_index = 0
end
whole_row = whole_row[mc_index + 3, whole_row.length - mc_index + 2]

#Skip EAC and CAC data at the front if there
if whole_row.include? 'CAC'
colon_index = whole_row.index('::')
if colon_index
whole_row = whole_row[colon_index + 1, whole_row.length - colon_index]
else
colon_index = whole_row.index(';;')
whole_row = whole_row[colon_index + 1, whole_row.length - colon_index]
end
if whole_row.include? 'EAC'
colon_index = whole_row.index('::')
if colon_index
whole_row = whole_row[colon_index + 1, whole_row.length - colon_index]
else
colon_index = whole_row.index(';;')
end

whole_row = whole_row[0, comma_index]
results = Array.new
ampersands_index = 0
next_ampersand_index = 0
while next_ampersand_index
results.push( get_single_knowledge_topic(whole_row[0, ampersands_index]) )
whole_row = whole_row[ampersands_index + 2, whole_row.length - (ampersands_index+2) ]
ampersands_index = ampersands_index + 2
next_ampersand_index = whole_row.index('&&', ampersands_index + 1)
end

return KnowledgeTopic.find_or_create_by(knowledge_topic: 'Unclassified')
ampersands_index = whole_row.index('&&')

results.push(get_single_knowledge_topic(whole_row))

return results

def self.get_single_knowledge_topic(title)

slash_index = title.index('/')

if slash_index.eql? nil
    return KnowledgeTopic.find_or_create_by(knowledge_topic: title)
end

ka_sloppy = title[0, slash_index]

if ka_sloppy[0, 3].include? 'MATH'
    ka_sloppy = 'MATH'
end

first_letter = ka_sloppy[0]

if first_letter >= 'A' && first_letter <= 'Z'
    ka_sloppy = ka_sloppy[1, 300]
end

kt = title[slash_index + 1, 300]

ku = ary[0]

kt = ary[1]

unless kt == nil
    KnowledgeTopic.where(knowledge_area: ka, knowledge_unit: ku).find_each do |item|
        temp_kt = item.knowledge_topic
        unless temp_kt == nil
            temp_kt = temp_kt.to_s
            if temp_kt.include?(kt)
                return item
            end
        end
    end
end

k = KnowledgeTopic.create(knowledge_topic: kt, knowledge_unit: ku, knowledge_area: ka)
k.save

return k

end

def self.clean_question(question_title)

question_title = question_title.gsub(/<[^>]*>/, '')

question_title = question_title.strip[0, 255]

return question_title
end

def self.load_answer(question_title, question_answer, enrolled, correct, kts)

question_text = clean_question(question_title)

ku = ary[0]

kt = ary[1]

end

unless kt == nil
    KnowledgeTopic.where(knowledge_area: ka, knowledge_unit: ku).find_each do |item|
        temp_kt = item.knowledge_topic
        unless temp_kt == nil
            temp_kt = temp_kt.to_s
            if temp_kt.include?(kt)
                return item
            end
        end
    end
end

k = KnowledgeTopic.create(knowledge_topic: kt, knowledge_unit: ku, knowledge_area: ka)
k.save

return k
end

def self.clean_question(question_title)

question_title = question_title.gsub(/<[^>]*>/, '')

question_title = question_title.strip[0, 255]

return question_title
end

def self.load_answer(question_title, question_answer, enrolled, correct, kts)

question_text = clean_question(question_title)
Answer.find_or_create_by(enroll_id: enrolled.id, question_text: question_text)
a.answer = question_answer
a.correct = correct
a.save

kts.each do |kt|
  if correct
    if kt.correct_answers
      kt.correct_answers += 1
    else
      kt.correct_answers = 1
    end
  else
    if kt.incorrect_answers
      kt.incorrect_answers += 1
    else
      kt.incorrect_answers = 1
    end
  end
  kt.save
end

KtsCoveredByAnswer.find_or_create_by(knowledge_topic: kt, answer: a)

self.load_demographics(enrolled, first_name, last_name, gender, academic_progress, ethnicity, student)
student.first_name = first_name
student.last_name = last_name
gender = gender.gsub(/<[^>]*>/, '')
student.gender = gender
academic_progress = academic_progress.gsub(/<[^>]*>/, '')
enrolled.academic_progress = academic_progress
ethnicity = ethnicity.gsub(/<[^>]*>/, '')
student.ethnicity = ethnicity
unless student.path_1160
  student.path_1160 = false
end
student.save
enrolled.save
end

self.load_program_of_study(programs_of_study, enrolled)

programs_of_study.each do |current_program|
  ProgramOfStudy.find_or_create_by(enroll_id: enrolled.id, program: current_program)
end

def self.clean_course_string(course)
  if course.include? 'CS'
    index = course.index('CS')
    course = course[index, 7]
    unless course.include? ' '
      course = course.chomp
    end
  elsif course.include? 'CEG'
    index = course.index('CEG')
    course = course[index, 8]
    unless course.include? ' '
      course = course.chomp
    end
  elsif course.include? 'MTH'
    index = course.index('MTH')
    course = course[index, 8]
    unless course.include? ' '
      course = course.chomp
    end
  elsif course.include? 'EGR'
    index = course.index('EGR')
    course = course[index, 6]
    unless course.include? ' '
      course = course.chomp
    end
  end
  course
end
index = course.index('EGR')
course = course[index, 8] unless course.include? ' '
course = course.chop end
elsif course.include? 'other'
course = 'other'
else
course = 'NONE'
end
return course end

def self.clean_program_from_string(course)
    program = course
    if course.include? 'CS'
        program = 'CS'
    elsif course.include? 'CEG'
        program = 'CEG'
    elsif course.include? 'MTH'
        program = 'MTH'
    elsif course.include? 'EGR'
        program = 'EGR'
    end
    return program end

def self.clean_number_from_string(course)
    number = 9999
    if course.include? 'CS'
        parts = course.partition('CS')
        number = parts[2].strip.to_i
    elsif course.include? 'CEG'
        parts = course.partition('CEG')
        number = parts[2].strip.to_i
    elsif course.include? 'MTH'
        parts = course.partition('MTH')
        number = parts[2].strip.to_i
    elsif course.include? 'EGR'
        parts = course.partition('EGR')
        number = parts[2].strip.to_i
    end
    return number end

def self.clean_semester(semester)
    if semester.include? 'Summer'
        semester = 'Summer'
    elsif semester.include? 'Spring'
        semester = 'Spring'
    else
        semester = 'Fall'
    end
    return semester end

def self.clean_pre_course(pre_course)
    pre_course = clean_course_string(pre_course)
    program = clean_program_from_string(pre_course)
    number = clean_number_from_string(pre_course)
    course = program + ' ' + number.to_s
    return course end

def self.clean_instructor(name)
    unless name == nil
        name = name.gsub(/<[^>]*>/, '')
        if name.include? '
            temp_ary = name.split(' ')
            return temp_ary[temp_ary.length - 1].strip
        end
    end
    return name
return ''
end
575
def
self.clean_question_answer(text)
text = text.gsub(/<[a-z]>/, '')
text = text.gsub(/<\/[a-z]>/, '')
text = text.gsub(/&nbsp/, '')
text = text[0, 250]
return text
end
end
585 586 587
class LoadStudentLearningOutcome
def self.import_slo_data(file)
  sheet1 = open_spreadsheet(file)
  index = 2 # Skip row of headers
  number_of_lines = sheet1.last_row
  # loop for each student in the file
  while index < number_of_lines do
    row = sheet1.row(index)
    slo = StudentLearningOutcome.find_or_create_by(accreditation_body: row[0], title: row[1])
    slo.description = row[2]
    slo.active = true
    unless slo.year_added
      slo.year_added = row[3]
    end
    slo.save
    index = index + 1
  end
end

def self.import_title_data(file)
  fileObj = File.new(file.path, "r")
  while (title = fileObj.gets)
    title = clean_title(title)
    question = fileObj.gets
    kts = load_knowledge_topic(title)
    question = clean_question(question)
    answers = Answer.where(question_text: question)
    kts.each do |kt|
      answers.each do |answer|
        item = KtsCoveredByAnswer.find_or_create_by(answer: answer, knowledge_topic: kt)
        item.save
      end
    end
  end
  fileObj.close
end

def self.open_spreadsheet(file)
  case File.extname(file.original_filename)
  when '.csv' then Roo::Csv.new(file.path)
  when '.xls' then Roo::Excel.new(file.path)
  when '.xlsx' then Roo::Excelx.new(file.path)
  else
    raise "Unknown file type: #{file.original_filename}"
  end
end

def self.import_title_data(file)
  fileObj = File.new(file.path, "r")
  while (title = fileObj.gets)
    title = clean_title(title)
    question = fileObj.gets
    kts = load_knowledge_topic(title)
    question = clean_question(question)
    answers = Answer.where(question_text: question)
    kts.each do |kt|
      answers.each do |answer|
        item = KtsCoveredByAnswer.find_or_create_by(answer: answer, knowledge_topic: kt)
        item.save
      end
    end
  end
  fileObj.close
end

def self.clean_question(question_title)
  if question_title
    question_title = question_title.gsub(/<[^>]*>/, '')
    question_title = question_title.strip[0, 255]
  end
  return question_title
end

else
  raise "Unknown file type: #{file.original_filename}"
end
end

36
def self.import_title_data(file)
  fileObj = File.new(file.path, "r")
  while (title = fileObj.gets)
    title = clean_title(title)
    question = fileObj.gets
    kts = load_knowledge_topic(title)
    question = clean_question(question)
    answers = Answer.where(question_text: question)
    kts.each do |kt|
      answers.each do |answer|
        item = KtsCoveredByAnswer.find_or_create_by(answer: answer, knowledge_topic: kt)
        item.save
      end
    end
  end
  fileObj.close
end

49
def self.clean_question(question_title)
  if question_title
    question_title = question_title.gsub(/<[^>]*>/, '')
    question_title = question_title.strip[0, 255]
  end
  return question_title
end

292
def self.clean_title(question_title)
    question_title = question_title.gsub(/<[^>]*>/, '')
    return question_title
end

def self.load_knowledge_topic(row)
    # Skip EAC and CAC data at the front if there
    if row.include? 'CAC'
        colon_index = row.index(':')
        if colon_index
            row = row[colon_index + 1, row.length - colon_index]
        else
            colon_index = row.index(';')
            row = row[colon_index + 1, row.length - colon_index]
        end
    end
    if row.include? 'EAC'
        colon_index = row.index(':')
        if colon_index
            row = row[colon_index + 1, row.length - colon_index]
        else
            colon_index = row.index(';')
            row = row[colon_index + 1, row.length - colon_index]
        end
    end
    if row.include? 'UNCLASSIFIED'
        return KnowledgeTopic.find_or_create_by(knowledge_topic: 'Unclassified')
    end

    results = Array.new
    ampersands_index = row.index('&&')
    while ampersands_index
        results.push(get_single_knowledge_topic(row[0, ampersands_index]))
        row = row[ampersands_index + 2, row.length - (ampersands_index + 2)]
        ampersands_index = row.index('&&')
    end
    return results
end

def self.get_single_knowledge_topic(title)
    slash_index = title.index('/')
    if slash_index.eql? nil
        return KnowledgeTopic.find_or_create_by(knowledge_topic: title)
    end
    ka_sloppy = title[0, slash_index]
    if ka_sloppy[0, 3].include? 'MATH'
        ka_sloppy = 'MATH'
    end
    first_letter = ka_sloppy[0]
unless first_letter >= 'A' && first_letter <= 'Z'
ka_sloppy = ka_sloppy[1, 300]
end

kt = title[slash_index + 1, 300]
ka = ka_sloppy
ary = kt.split(':'
if ary.length > 1
ku = ary[0]
kt = ary[1]
end

kt = kt.gsub(/[;:]/, '').gsub(/\(.*/, '').strip

kt = kt[0, 250]

unless kt == nil

KnowledgeTopic.where(knowledge_area: ka).order(:id).find_each

do |item|
  temp_kt = item.knowledge_topic
  unless temp_kt == nil
    temp_kt = temp_kt.to_s
    if temp_kt.include?(kt)
      return item
    end
  end
end

end

k = KnowledgeTopic.create(knowledge_topic: kt, knowledge_unit: ku, knowledge_area: ka)
k.save

return k
end
File - D:\railsApps\playgroundJan2016\ app\controllers\answers_controller.rb

class AnswersController < ApplicationController
  before_action :set_answer,
  only: [:show, :edit, :update, :destroy]

  # GET /answers
  # GET /answers.json
  def index
    @answers = Answer.all
  end

  # GET /answers/1
  # GET /answers/1.json
  def show
  end

  # GET /answers/new
  def new
    @answer = Answer.new
  end

  # GET /answers/1/edit
  def edit
  end

  # POST /answers
  # POST /answers.json
  def create
    @answer = Answer.new(answer_params)
    respond_to do |format|
      if @answer.save
        format.html { redirect_to @answer, notice: 'Answer is successfully created.' }
        format.json { render :show, status: :created, location: @answer }
      else
        format.html { render :new }
        format.json { render json: @answer.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /answers/1
  # PATCH/PUT /answers/1.json
  def update
    respond_to do |format|
      if @answer.update(answer_params)
        format.html { redirect_to @answer, notice: 'Answer is successfully updated.' }
        format.json { render :show, status: :ok, location: @answer }
      else
        format.html { render :edit }
        format.json { render json: @answer.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /answers/1
  # DELETE /answers/1.json
  def destroy
    @answer.destroy
    respond_to do |format|
  end
end
format.html { redirect_to answers_url, notice: 'Answer is successfully destroyed.' }
format.json { head :no_content }
end

private
# Use callbacks to share common setup or constraints between actions.
def set_answer
  @answer = Answer.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def answer_params
end

Page 2 of 2
File
D:\railsApps\playgroundJan2016\app\controllers\enrolls_controller.rb
class EnrollsController < ApplicationController
  before_action :set_enroll, only: [:show, :edit, :update, :destroy]
  # GET /enrolls
  # GET /enrolls.json
  def index
    @enrolls = Enroll.all
  end

  # GET /enrolls/1
  # GET /enrolls/1.json
  def show
    end

14  # GET /enrolls/new
15  def new
16    @enroll = Enroll.new
17  end
19  # GET /enrolls/1/edit
20  def edit
23  # POST /enrolls
24  # POST /enrolls.json
25  def create
28    respond_to do |format|
29      if @enroll.save
30        format.html { redirect_to @enroll, notice: 'Enroll is successfully created.' }
31        format.json { render :show, status: :created, location: @enroll }
32      else
33        format.html { render :new }
34        format.json { render json: @enroll.errors, status: :unprocessable_entity }
35    end
39  end
43  # PATCH/PUT /enrolls/1
44  # PATCH/PUT /enrolls/1.json
45  def update
48    respond_to do |format|
49      if @enroll.update(enroll_params)
50        format.html { redirect_to @enroll, notice: 'Enroll is successfully updated.' }
51        format.json { render :show, status: :ok, location: @enroll }
52      else
53        format.html { render :edit }
54      end
56    end
}
format.json { render json: @enroll.errors, status: :unprocessable_entity }

# DELETE /enrolls/1
# DELETE /enrolls/1.json
def destroy
  @enroll.destroy
  respond_to do |format|
    format.html { redirect_to enrolls_url, notice: 'Enroll is successfully destroyed.' }
    format.json { head :no_content }
  end
end

private

# Use callbacks to share common setup or constraints between actions.
def set_enroll
  @enroll = Enroll.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def enroll_params
  params.require(:enroll).permit(
    :Student_id,
    :course_registration_number,
    :academic_progress, :grade, :
    quiz_score, :year_offered, :
    semester, :section,
    :faculty_last_name,
    :faculty_first_name, :course,
    :teaching_style)
end

Definitely looks like some code is mixed up here. It seems there's a mix of different classes and controllers with varying levels of completeness. The code snippets I can make sense of are:

```ruby
# DELETE /enrolls/1
# DELETE /enrolls/1.json
def destroy
  @enroll.destroy
  respond_to do |format|
    format.html { redirect_to enrolls_url, notice: 'Enroll is successfully destroyed.' }
    format.json { head :no_content }
  end
end

private

# Use callbacks to share common setup or constraints between actions.
def set_enroll
  @enroll = Enroll.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def enroll_params
  params.require(:enroll).permit(
    :Student_id,
    :course_registration_number,
    :academic_progress, :grade, :
    quiz_score, :year_offered, :
    semester, :section,
    :faculty_last_name,
    :faculty_first_name, :course,
    :teaching_style)
end
```

It seems like there's a mix of Control file names, directories, and file references which could indicate that the code is part of a larger application or project, possibly a Rails application given the file structure and naming conventions. The code appears to be related to handling enrollments and destruction of enrollments, with some internal logic to ensure that only allowed parameters are processed.
class StudentsController < ApplicationController
  before_action :set_student,
  only: [:show, :edit, :update, :destroy]

  # GET /students
  # GET /students.json
  def index
    @students = Student.all
  end

  # GET /students/1
  # GET /students/1.json
  def show
  end

  # GET /students/new
  def new
    @student = Student.new
  end

  # GET /students/1/edit
  def edit
  end

  # POST /students
  # POST /students.json
  def create
    @student = Student.new(student_params)
    respond_to do |format|
      if @student.save
        format.html { redirect_to @student, notice: 'Student is successfully created.' }
        format.json { render :show, status: :created, location: @student }
      else
        format.html { render :new }
        format.json { render json: @student.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /students/1
  # PATCH/PUT /students/1.json
  def update
    respond_to do |format|
      if @student.update(student_params)
        format.html { redirect_to @student, notice: 'Student is successfully updated.' }
        format.json { render :show, status: :ok, location: @student }
      else
        format.html { render :edit }
        format.json { render json: @student.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /students/1
  # DELETE /students/1.json
  def destroy
    @student.destroy
    respond_to do |format|
      format.html { redirect_to students_url, notice: 'Student is successfully destroyed.' }
      format.json { head :no_content }
    end
  end

private

app\controllers\students_controller.rb

format.json { render json: @student.errors, status: :unprocessable_entity }
end
end
end
39
# PATCH/PUT /students/1
# PATCH/PUT /students/1.json
def update
  respond_to do |format|
    if @student.update(student_params)
      format.html { redirect_to @student, notice: 'Student is successfully updated.' }
      format.json { render :show, status: :ok, location: @student }
    else
      format.html { render :edit }
      format.json { render json: @student.errors, status: :unprocessable_entity }
    end
  end
end
end
53
# DELETE /students/1
# DELETE /students/1.json
def destroy
  @student.destroy
  respond_to do |format|
    format.html { redirect_to students_url, notice: 'Student is successfully destroyed.' }
    format.json { head :no_content }
  end
end
end
63
private
# Use callbacks to share common setup or constraints between actions.
def set_student
@student = Student.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def student_params
end

delete /slo_averages/1
# DELETE /slo_averages/1.json
def destroy
@slo_average.destroy
respond_to do |format|
  format.html { redirect_to slo_averages_url, notice: 'Slo average is successfully destroyed.' }
  format.json { head :no_content }
end

private
# Use callbacks to share common setup or constraints between actions.
def set_slo_average
@slo_average = SloAverage.find(params[:id])
end

# DELETE /slo_averages/1
# DELETE /slo_averages/1.json
def destroy
@slo_average.destroy
respond_to do |format|
  format.html { redirect_to slo_averages_url, notice: 'Slo average is successfully destroyed.' }
  format.json { head :no_content }
end

private
# Use callbacks to share common setup or constraints between actions.
def set_slo_average
@slo_average = SloAverage.find(params[:id])
end

# Prevent CSRF attacks by raising an exception.
# For APIs, you may want to use :null_session instead.
protect_from_forgery with: :exception
end

File
D:\railsApps\playgroundJan2016\app\controllers\slo_averages_controller.rb
1 class PrerequisitesController < ApplicationController
2   before_action :set_prerequisite, only: [:show, :edit, :update, :destroy]
end
end
end
53
def index
@prerequisites = Prerequisite.all
end

# GET /prerequisites/1
# GET /prerequisites/1.json
def show
end

# GET /prerequisites/new
def new
@prerequisite = Prerequisite.new
end

# GET /prerequisites/1/edit
def edit
end

# POST /prerequisites
# POST /prerequisites.json
def create
@prerequisite = Prerequisite.new(prerequisite_params)
respond_to do |format|
  if @prerequisite.save
    format.html { redirect_to @prerequisite, notice: 'Prerequisite is successfully created.' }
    format.json { render :show, status: :created, location: @prerequisite }
  else
    format.html { render :new }
    format.json { render json: @prerequisite.errors, status: :unprocessable_entity }
  end
end

# PATCH/PUT /prerequisites/1
# PATCH/PUT /prerequisites/1.json
def update
respond_to do |format|
  @prerequisite.update(prerequisite_params)
  @prerequisite, notice: 'Prerequisite is successfully updated.' }
  status: :ok, location: @prerequisite }
  @prerequisite.errors, status: :unprocessable_entity }
end
end
end

# DELETE /prerequisites/1
# DELETE /prerequisites/1.json
def destroy
@prerequisite.destroy
respond_to do |format|
  format.html { redirect_to prerequisites_url, notice: 'Prerequisite is successfully destroyed.' }
  format.json { head :no_content }
end
end
end

private

# Use callbacks to share common setup or constraints between actions.
def set_prerequisite
  @prerequisite = Prerequisite.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def prerequisite_params
params.require(:prerequisite).permit(:course, :Enroll_id, :year_taken, :faculty_last_name, :semester)
end
end

File
D:\railsApps\playgroundJan2016\app\controllers\class_averages_controller.rb

class ClassAveragesController < ApplicationController
before_action :set_class_average, only: [:show, :edit, :update, :destroy]
3
# GET /class_averages
# GET /class_averages.json
def index
@class_averages = ClassAverage.all
end

# GET /class_averages/1
# GET /class_averages/1.json
def show
end

# GET /class_averages/new
def new
@class_average = ClassAverage.new
end

# GET /class_averages/1/edit
def edit
end
23
# POST /class_averages
# POST /class_averages.json
def create

@class_average = ClassAverage.new(class_average_params)
28
respond_to do |format|
  if @class_average.save
    format.html { redirect_to @class_average, notice: 'Class average is successfully created.' }
    format.json { render :show, status: :created, location: @class_average }
  else
    format.html { render :new }
    format.json { render json: @class_average.errors, status: :unprocessable_entity }
  end
end

# PATCH/PUT /class_averages/1
# PATCH/PUT /class_averages/1.json
def update
  respond_to do |format|
    @class_average.update(class_average_params)
    format.html { render :show, status: :ok, location: @class_average }
    format.json { render json: @class_average.errors, status: :unprocessable_entity }
  end
end

# DELETE /class_averages/1
# DELETE /class_averages/1.json
def destroy
@class_average.destroy
respond_to do |format|
  format.html { redirect_to class_averages_url, notice: 'Class average is successfully destroyed.' }
  format.json { head :no_content }
end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_class_average
@class_average = ClassAverage.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def class_average_params
  params.require(:class_average).permit(:program, :course, :semester, :year_offered, :number_students, :quiz_average)
end

File - D:\railsApps\playgroundJan2016\app\controllers\knowledge_topics_controller.rb
class KnowledgeTopicsController < ApplicationController
  before_action :set_knowledge_topic, only: [:show, :edit, :update, :destroy]
end

# GET /knowledge_topics
# GET /knowledge_topics.json
def index

@knowledge_topics = KnowledgeTopic.all

# GET /knowledge_topics/1
# GET /knowledge_topics/1.json
def show
end

# GET /knowledge_topics/new
def new
  @knowledge_topic = KnowledgeTopic.new
end

# GET /knowledge_topics/1/edit
def edit
end

# POST /knowledge_topics
# POST /knowledge_topics.json
def create
  @knowledge_topic = KnowledgeTopic.new(knowledge_topic_params)
  respond_to do |format|
    if @knowledge_topic.save
      format.html { redirect_to @knowledge_topic, notice: 'Knowledge topic is successfully created.' }
      format.json { render :show, status: :created, location: @knowledge_topic }
    else
      format.html { render :new }
      format.json { render json: @knowledge_topic.errors, status: :unprocessable_entity }
    end
  end
end

# PATCH/PUT /knowledge_topics/1
# PATCH/PUT /knowledge_topics/1.json
def update
end

end

# PATCH/PUT
/knowledge_topics/1.json

def update
  respond_to do |format|
    @knowledge_topic.update(knowledge_topic_params)
    @knowledge_topic, notice: 'Knowledge topic is successfully updated.'
  end
  status: :ok,
  location: @knowledge_topic
end

private
  def set_knowledge_topic
    @knowledge_topic = KnowledgeTopic.find(params[:id])
  end
end

# DELETE /knowledge_topics/1
# DELETE /knowledge_topics/1.json

def destroy
  @knowledge_topic.destroy
  respond_to do |format|
    format.html { redirect_to knowledge_topics_url, notice: 'Knowledge topic is successfully destroyed.' }
    format.json { head :no_content }
  end
end

class StandardReportsController
  < ApplicationController

  def index
  end

  def answers
  end

  def search
    @answers = Answer.select(:answer, :correct, 'count(answers.id) AS number_students_selected').group(:answer)
    @correct = @answers.where(correct: true).select(:correct, 'count(answers.id) AS number').group(:correct).first
     @incorrect = @answers.where(correct: false).select(:correct, 'count(answers.id) AS number').group(:correct).first
    render '/standard_report/answers'
  end

  def abet_repo
end

end
@knowledge_areas_all = StandardReport.get_knowledge_areas()
@knowledge_units_all = StandardReport.get_knowledge_units()
@knowledge_topics_all = StandardReport.get_knowledge_topics()
@student_learning_outcomes_all = StandardReport.get_student_learning_outcomes()
@different_programming_path = StandardReport.get_programming_path_divide()
22
@knowledge_areas_year = StandardReport.combine_academic_years(@knowledge_areas_all, 3, 3)
@knowledge_units_year = StandardReport.combine_academic_years(@knowledge_units_all, 4, 3)
@knowledge_topics_year = StandardReport.combine_academic_years(@knowledge_topics_all, 4, 3)
@student_learning_outcomes_year = StandardReport.combine_academic_years(@student_learning_outcomes_all, 3, 3)
@different_programming_path_years = StandardReport.combine_academic_years(@different_programming_path, 4, 6)
28
@student_learning_outcomes_year_percentages = StandardReport.find_percentages_years(@student_learning_outcomes_year, 1, 2, 4, 5)
31
@student_learning_outcomes_all_percentages = StandardReport.find_percentages_term(@student_learning_outcomes_all, 0, 1, 2, 4, 5)
@student_learning_outcomes_year_percentages_transposed = StandardReport.transpose_percentage_array(@student_learning_outcomes_year_percentages)
34
@slos_stackable_years = StandardReport.get_category_stackables(@student_learning_outcomes_year, 1, 2, 4, 3)
@ka_stackable_years = StandardReport.get_category_stackables(@knowledge_areas_year, 1, 2, 4, 3)
37
38
@different_programming_path_slos = StandardReport.different_paths_slos()
@different_programming_path_grades = StandardReport.different_paths_grades()
@different_programming_path_grades_core = StandardReport.reform_table(@different_programming_path_grades)
@cs_1181_students_from_1160 = StandardReport.list_1160_students('CS', 1181)
@ceg_3310_students_from_1160 = StandardReport.list_1160_students('CEG', 3310)
@cs_3100_students_from_1160 = StandardReport.list_1160_students('CS', 3100)
render '/standard_reports/abet_report' end
46
def quiz_coverage
@slo_coverage_cs = StandardReport.slo_coverage_for_cs
@slo_coverage_ceg = StandardReport.slo_coverage_for_ceg
@cs_slos = StudentLearningOutcome.where(accreditation_body: 'CAC')
@ceg_slos = StudentLearningOutcome.where(accreditation_body: 'EAC')
render '/standard_reports/quiz_coverage' end
57
def summary_report
@semester = 'Spring'
#StandardReport.get_semester()
@year_offered = 2016
#StandardReport.get_year_offered()
@course_averages = ClassAverage.where(semester: @semester, @year_offered).order(:course)
@current_averages = StandardReport.get_current_course_averages(@semester, @year_offered)
@course_table = StandardReport.get_course_boxplot_data(@semester, @year_offered)
@course_averages = StandardReport.create_course_summary_table(@course_averages, @year_offered, @semester)
66
@slo_averages = StandardReport.get_slo_averages(@semester, @year_offered).order(:program, :title)
@current_slo_averages = @slo_averages.pluck(:average)
@slo_table = StandardReport.get_slo_table(@semester, @year_offered)
@slo_averages = StandardReport.create_slo_summary_table(@slo_averages, @year_offered, @semester) 71
@courses_to_breakdown = StandardReport.get_items_to_breakdown(@course_averages)
@courses_breakdown = StandardReport.get_course_breakdown(@courses_to_breakdown, @semester, @year_offered)
74
@slos_to_breakdown = StandardReport.get_items_to_breakdown(@slo_averages)
@slos_breakdown = StandardReport.get_slo_breakdown(@slos_to_breakdown, @semester, @year_offered) 77
render '/standard_reports/summary_report' end
80
def quiz_results_by_class
year_needed = [2013, 2014]
course = 'CS 1181'
semester = 'Fall'
all_students_for_course = Enroll.where(semester: semester, course: course, year_offered: year_needed)  
answers = all_students_for_course.joins(:answers).select('enrolls.*,
answers.*')  
@results = answers.select('enrolls.student_id AS w_number', :correct,
:question_text)  
render '/standard_reports/quiz_results_by_class'  
def pedagogy_comparison  
old_pedagogy = ['Fall 2012', 'Spring 2013', 'Summer 2013', 'Fall 2013', 'Spring 2014', 'Summer 2014', 'Fall 2014']  
new_pedagogy = ['Spring 2015', 'Summer 2015', 'Fall 2015']  
@students = StandardReport.get_pedagogy_comparison('CS', 1180, old_pedagogy, new_pedagogy)  
render '/standard_reports/pedagogy_comparison'  
def single_course_report  
@student_list = StandardReport.get_student_list('CS', 1180, 'Spring', 2015, 'Doom')  
@students = Student.where(w_number: ['w001mgu', 'w001ojp', 'w001xnw',
'w003nce', 'w005taz',
'w006jmi', 'w007avc', 'w007bhh',
'w007ehs', 'w007wxb', 'w008erj',
'w014ccg', 'w014mww', 'w017emj',
'w018ljw',
'w025akc', 'w025txf', 'w028pjh',
'w030jrj', 'w033rnx', 'w035saf',
'w052djc', 'w053djc', 'w061dxl',
'w075bjb',
'w086tmm', 'w117rts', 'w353jar',
'w737jms']).pluck(:id)  
# Starkey  
render '/standard_reports/quiz_results_by_class'  
def pedagogy_comparison  
old_pedagogy = ['Fall 2012', 'Spring 2013', 'Summer 2013', 'Fall 2013', 'Spring 2014', 'Summer 2014', 'Fall 2014']  
new_pedagogy = ['Spring 2015', 'Summer 2015', 'Fall 2015']  
@students = StandardReport.get_pedagogy_comparison('CS', 1180, old_pedagogy, new_pedagogy)  
render '/standard_reports/pedagogy_comparison'  
def single_course_report  
@student_list = StandardReport.get_student_list('CS', 1180, 'Spring', 2015, 'Doom')  
# Doom  
@student_list = StandardReport.get_student_list('CS', 1180, 'Spring', 2015, 'Doom')  
# Control  
# @students = Student.where(w_number: ['w003whw', 'w021acd', 'w007cht',
'w001ijr', 'w056mjr',
'w012bwh', 'w043rap', 'w046cam',
'w049jkb', 'w147mcs', 'w019cea',
'w039pam', 'w013etr', 'w004twj',
'w023ddj',
'w031tml', 'w019nrcs', 'w336maw',
'w019ead', 'w262jxi', 'w020jtj',
'w003dhp', 'w033kch', 'w075bam',
'w058jmm',
'w030jcs', 'w001ldi', 'w001tdc',
'w002drz', 'w028krd', 'w011wjk',
'w057dmr', 'w013teg', 'w596mam',
'w099bas',
'w025sha', 'w018aoa', 'w023mra',
'w017dwb', 'w393sxc', 'w107jtc',
'w030bjd', 'w011bgh', 'w004hth',
'w014tci',
'w493amm', 'w006vdp', 'w022rcr',
'w077das', 'w044abs', 'w090mms',
'w401jxs', 'w004cpt', 'w040dxt',
'w008mtv', 'w096alw']).pluck(:id)  
# Control  
# @students = Student.where(w_number: ['w003whw', 'w021acd', 'w007cht',
'w001ijr', 'w056mjr',
'w012bwh', 'w043rap', 'w046cam',
'w049jkb', 'w147mcs', 'w019cea',
'w039pam', 'w013etr', 'w004twj',
'w023ddj',
'w031tml', 'w019nrcs', 'w336maw',
'w019ead', 'w262jxi', 'w020jtj',
'w003dhp', 'w033kch', 'w075bam',
'w058jmm',
'w030jcs', 'w001ldi', 'w001tdc',
'w002drz', 'w028krd', 'w011wjk',
'w057dmr', 'w013teg', 'w596mam',
'w099bas',
'w025sha', 'w018aoa', 'w023mra',
'w017dwb', 'w393sxc', 'w107jtc',
'w030bjd', 'w011bgh', 'w004hth',
'w014tci',
'w493amm', 'w006vdp', 'w022rcr',
'w077das', 'w044abs', 'w090mms',
'w401jxs', 'w004cpt', 'w040dxt',
'w008mtv', 'w096alw']).pluck(:id)
\[
\text{\texttt{\begin{verbatim}
"w416jmw", "w006stk", \\
"w463jlc", "w003zld", \\
w078lm", "w027bjd", "w025nak", \\
w070cjh", "w005ary", \\
w002vij", "w082mm", \\
w065rm", "w069rxg", \\
w029crr", "w049dsm", "w078djm", \\
w055adc", "w011rng", \\
w037bjc", "w022jcd", \\
w276axb", "w004rgt", \\
w031tal", "w025naw", "w007nsq", \\
w037mdk", "w022jxe", \\
w07cck", "w022jfs", \\
w055sjb", "w442jaw", \\
w014stc", "w090cxh", "w032jng", \\
w043rc", "w002oxc", \\
w010ckh", "w073mar", \\
w006rth", "w110dhh", \\
w002mpi", "w065jms", "w012agw", \\
w048sb", "w142bcb", \\
w038ara", "w047jcr", \\
w014kxe", "w073mr", "w425mah", \\
w004ht", "w052vp", \\
w034alt", "w020aae", \\
w034xb", "w059jdt", \\
w006zfa", "w041ja", "w128jda", \\
w014nkb", "w010apl", \\
w061mjr", "w028jtg", \\
w010wdh", "w102stk", "w064ka", \\
"w067bmm", "w028nds", \\
w018mps", "w035bat", \\
w034aw", "w041jmn", \\
w066hma", "w019ae", \\
w001epv", \\
w065seb", "w016sja", \\
w012rle", "w425mah", \\
w066bmm", "w008ims", \\
w049smt", "w045aha"
\end{verbatim}}
\) }
\]
end
112
113 def abet_data_collection_summary_course_focus
114 @terms = StandardReport.get_array_of_terms_for_last_five_years()
115 @courses_with_questions = StandardReport.courses_knowledge_topics_slo_summary_results(@terms)
117 118 end
119
120 def abet_data_collection_summary_slo_focus
121 @cac_slos = StudentLearningOutcome.where(accreditation_body: 'CAC').order(:title).pluck(:title)
122 @eac_slos = StudentLearningOutcome.where(accreditation_body: 'EAC').order(:title).pluck(:title)
124 @terms = StandardReport.get_array_of_terms_for_last_five_years()
125 @slos_with_course_and_kt = StandardReport.courses_knowledge_topics_slo_summary_results(@terms)
127 128 end
129
130 131 end
132
class ProgramOfStudiesController < ApplicationController
  before_action :set_program_of_study, only: [:show, :edit, :update, :destroy]

  # GET /program_of_studies
  # GET /program_of_studies.json
  def index
    @program_of_studies = ProgramOfStudy.all
  end

  # GET /program_of_studies/1
  # GET /program_of_studies/1.json
  def show
  end

  # GET /program_of_studies/new
  def new
    @program_of_study = ProgramOfStudy.new
  end

  # GET /program_of_studies/1/edit
  def edit
  end

  # POST /program_of_studies
  # POST /program_of_studies.json
  def create
    @program_of_study = ProgramOfStudy.new(program_of_study_params)
    respond_to do |format|
      if @program_of_study.save
        format.html { redirect_to @program_of_study, notice: 'Program of study is successfully created.' }
        format.json { render :show, status: :created, location: @program_of_study }
      else
        format.html { render :new }
        format.json { render json: @program_of_study.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /program_of_studies/1
  # PATCH/PUT /program_of_studies/1.json
  def update
    respond_to do |format|
      @program_of_study.update(program_of_study_params)
      @program_of_study, notice: 'Program of study is successfully updated.' }
      status: :ok, location: @program_of_study }
    end
  end

  # DELETE /program_of_studies/1
  # DELETE /program_of_studies/1.json
  def destroy
    @program_of_study.destroy
    respond_to do |format|
      format.html { redirect_to program_of_studies_url, notice:
      format.json { render json: @program_of_study.errors, status: :unprocessable_entity }
    end
  end
end
Program of study is successfully destroyed.' }
format.json { head :no_content } end
end
63 private
# Use callbacks to share common setup or constraints between actions.
def set_program_of_study
@program_of_study = ProgramOfStudy.find(params[:id])
end
69
70 # Never trust parameters from the scary internet, only allow the white list through.
def program_of_study_params
params.require(:program_of_study).permit(:program, :Enroll_id)
end
75
File - D:\railsApps\playgroundJan2016\app\controllers\slo_covered_by_kts_controller.rb
class SloCoveredByKtsController < ApplicationController
before_action :set_slo_covered_by_kt, only: [:show, :edit, :update, :destroy]
3
# GET /slo_covered_by_kts
# GET /slo_covered_by_kts.json
def index
@slo_covered_by_kts = SloCoveredByKt.all
end
9
# GET /slo_covered_by_kts/1
# GET /slo_covered_by_kts/1.json
def show
end
14
# GET /slo_covered_by_kts/new
def new
@slo_covered_by_kt = SloCoveredByKt.new
end
19
# GET /slo_covered_by_kts/1/edit
def edit
end
23
# POST /slo_covered_by_kts
# POST /slo_covered_by_kts.json
def create
@slo_covered_by_kt = SloCoveredByKt.new(slo_covered_by_kt_params)
28
respond_to do |format|
if @slo_covered_by_kt.save
format.html { redirect_to @slo_covered_by_kt, notice: 'Slo covered by kt is successfully created.' }
format.json { render :show, status: :created, location: @slo_covered_by_kt }
else
format.html { render :new }
format.json { render json: @slo_covered_by_kt.errors, status: :unprocessable_entity }
end
end
39
# PATCH/PUT /slo_covered_by_kts/1
# PATCH/PUT /slo_covered_by_kts/1.json
def update
respond_to do |format|
@slo_covered_by_kt.update(slo_covered_by_kt_params)
end

@slo_covered_by_kt, notice: 'Slo covered by kt is successfully updated.' } } status: :ok, location: @slo_covered_by_kt } @slo_covered_by_kt.errors, status: :unprocessable_entity } File D:\railsApps\playgroundJan2016\ app\controllers\slo_covered_by_kts_controller.rb end end end 53 # DELETE /slo_covered_by_kts/1 # DELETE /slo_covered_by_kts/1.json def destroy @slo_covered_by_kt.destroy respond_to do |format| format.html { redirect_to slo_covered_by_kts_url, notice: 'Slo covered by kt is successfully destroyed.' } format.json { head :no_content } end end 63 private # Use callbacks to share common setup or constraints between actions. def set_slo_covered_by_kt @slo_covered_by_kt = SloCoveredByKt.find(params[:id]) end 69 70 # Never trust parameters from the scary internet, only allow the white list through. def slo_covered_by_kt_params params.require(:slo_covered_by_kt).permit(:StudentLearningOutcome_id, :KnowledgeTopic_id) end
respond_to do |format|
  if @peo_covered_by_slo.save
    format.html { redirect_to @peo_covered_by_slo, notice: 'Peo covered by slo is successfully created.' }
    format.json { render :show, status: :created, location: @peo_covered_by_slo }
  else
    format.html { render :new }
    format.json { render json: @peo_covered_by_slo.errors, status: :unprocessable_entity }
  end
end

# PATCH/PUT
/peo_covered_by_slos/1
# PATCH/PUT
/peo_covered_by_slos/1.json
def update
  respond_to do |format|
    @peo_covered_by_slo.update(peo_covered_by_slo_params)
    @peo_covered_by_slo, notice: 'Peo covered by slo is successfully updated.'
    status: :ok, location: @peo_covered_by_slo
    @peo_covered_by_slo.errors, status: :unprocessable_entity
  end
end

70  # Never trust parameters from the scary internet, only allow the white list through.
def peo_covered_by_slo_params
  params.require(:peo_covered_by_slo).permit(:StudentLearningOutcome_id, :ProgramEducationalObjective_id)
end
end

File
D:\railsApps\playgroundJan2016\app\controllers\load_internal_reports_controller.rb
class
  LoadInternalReportsController < ApplicationController
def index
end
end

53  # DELETE
/peo_covered_by_slos/1
# DELETE
/peo_covered_by_slos/1.json
def destroy
  @peo_covered_by_slo.destroy
  respond_to do |format|
    format.html { redirect_to peo_covered_by_slos_url, notice: 'Peo covered by slo is successfully destroyed.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_peo_covered_by_slo
  @peo_covered_by_slo = PeoCoveredBySlo.find(params[:id])
end

311
class KtsCoveredByAnswersController < ApplicationController
  before_action :set_kts_covered_by_answer, only: [:show, :edit, :update, :destroy]

  # GET /kts_covered_by_answers
  GET /kts_covered_by_answers.json
  def index
    @kts_covered_by_answers = KtsCoveredByAnswer.all
  end

  # GET /kts_covered_by_answers/1
  GET /kts_covered_by_answers/1.json
  def show
  end

  # GET /kts_covered_by_answers/new
  def new
    @kts_covered_by_answer = KtsCoveredByAnswer.new
  end

  # GET /kts_covered_by_answers/1/edit
  def edit
  end

  # POST /kts_covered_by_answers
  POST /kts_covered_by_answers.json
  def create
    @kts_covered_by_answer = KtsCoveredByAnswer.new(kts_covered_by_answer_params)
    respond_to do |format|
      if @kts_covered_by_answer.save
        format.html { redirect_to @kts_covered_by_answer, notice: 'Kts covered by answer is successfully created.' }
        format.json { render :show, status: :created, location: @kts_covered_by_answer }
      else
        format.html { render :new }
        format.json { render json: @kts_covered_by_answer.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /kts_covered_by_answers/1
  PATCH/PUT /kts_covered_by_answers/1.json
  def update
    respond_to do |format|
      if @kts_covered_by_answer.update(kts_covered_by_answer_params)
        format.html { redirect_to @kts_covered_by_answer, notice: 'Kts covered by answer is successfully updated.' }
        format.json { render :show, status: :ok, location: @kts_covered_by_answer }
      else
        format.html { render :edit }
        format.json { render json: @kts_covered_by_answer.errors, status: :unprocessable_entity }
      end
    end
  end
end
# DELETE /kts_covered_by_answers/1
# DELETE /kts_covered_by_answers/1.json

def destroy
  @kts_covered_by_answer.destroy
  respond_to do |format|
    format.html { redirect_to kts_covered_by_answers_url, notice: 'Kts covered by answer is successfully destroyed.' }
    format.json { head :no_content }
  end
end

private
  # Use callbacks to share common setup or constraints between actions.
  def set_kts_covered_by_answer
    @kts_covered_by_answer = KtsCoveredByAnswer.find(params[:id])
  end

  # Never trust parameters from the scary internet, only allow the white list through.
  def kts_covered_by_answer_params
    params.require(:kts_covered_by_answer).permit(:knowledge_topic_id, :answer_id)
  end
end

class LoadSloCoveredByKtsController < ApplicationController
  def index
    end
  end

  #render plain:
  params[:file].inspect
  LoadSloCoveredByKt.import_slo_covered_by_kt(params[:file])
  #LoadSloCoveredByKt.import_slo_covered_by_kt_ceg(params[:file])
  SloCoveredByKt.destroy_all
  render 'welcome/index'
end

end
#LoadStudentQuizResult.start_quiz_upload(params[:file], 10377, "CS", 3100, "Liu", 'Meilin', "Spring", 2014, 2)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 15393, "CS", 1181, "Ondrasek", 'Michael', "Spring", 2014, 90)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 11265, "CEG", 4350, "Chung", 'Soon', "Spring", 2014, 1)
46
47
#Summer 2014
#LoadStudentQuizResult.start_quiz_upload(params[:file], 40973, "CS", 1181, "Wlodarski", 'Daniel', "Summer", 2014, 1)

File
D:\railsApps\playgroundJan2016\app\controllers\load_student_quiz_results_controller.rb

#LoadStudentQuizResult.start_quiz_upload(params[:file], 45343, "CEG", 3310, "Doom", 'Travis', "Summer", 2014, 1)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 44706, "CS", 3100, "Buck", 'Eric', "Summer", 2014, 1)
  #Fall 2015
  ## means never got this data

#LoadStudentQuizResult.start_quiz_upload(params[:file], 73206, "CS", 3180, "Buck", 'Erik', "Fall", 2015, 1)

#LoadStudentQuizResult.start_quiz_upload(params[:file], 73196, "CS", 1181, "Volkers", 'Richard', "Fall", 2015, 1)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 73204, "CS", 3100, "Rizki", 'Matee n', "Fall", 2015, 1)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 79901, "CEG", 3310, "Raymer", 'Mic hael', "Fall", 2015, 2)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 73064, "CEG", 3310, "Jean", 'Jack', "Fall", 2015, 1)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 73069, "CEG", 3320, "Banerjee", 'Tanvi', "Fall", 2015, 1)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 73080, "CEG", 4350, "Chung", 'Soon', "Fall", 2015, 1)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 11242, "CEG", 2350, "Buck", 'Erik', "Spring", 2015, 1)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 11243, "CEG", 2350, "Kijowski", 'Matthew', "Spring", 2015, 2)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 11256, "CEG", 3320, "Banerjee", 'Tanvi', "Spring", 2015, 1)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 11251, "CEG", 3310, "Jean", 'Jack', "Spring", 2015, 2)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 10378, "CS", 3100, "Liu", 'Meilin', "Spring", 2015, 2)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 11358, "CS", 1181, "Ondrasek", 'Mich ael', "Spring", 2015, 1)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 11265, "CEG", 4350, "Chung", 'Soon ', "Spring", 2015, 1)
74
75
#Summer 2015
#LoadStudentQuizResult.start_quiz_upload(params[:file], 40973, "CS", 1181, "Timmerman", 'K athleen', "Summer", 2015, 1)
78
#Spring 2016
#LoadStudentQuizResult.start_quiz_upload(params[:file], 11242, "CEG", 2350, "Buck", 'Erik', "Spring", 2016, 1)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 11250, "CEG", 3310, "Banerjee", 'Tanvi', "Spring", 2016, 1)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 11251, "CEG", 3310, "Jean", 'Jack', "Spring", 2016, 2)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 11256, "CEG", 3320, "Doom", 'Travi s', "Spring", 2016, 1)
##LoadStudentQuizResult.start_quiz_upload(params[:file], 10358, "CS", 1181, "Volkers", 'Richard', "Spring", 2016, 1)
#LoadStudentQuizResult.start_quiz_upload(params[:file], 10377, "CS", 3100, "Raymer", 'Mich ael', "Spring", 2016, 1)
def index
@student_learning_outcomes = StudentLearningOutcome.all
end

9
#
GET
/student_learning_outcomes/1
#
GET
/student_learning_outcomes/1.json

def show
def edit
end

23
#
POST
/student_learning_outcomes
#
POST
/student_learning_outcomes.json

def create
@student_learning_outcome = StudentLearningOutcome.new(student_learning_outcome_params)
28
respond_to do |format|
  if
    @student_learning_outcome.save
      format.html { redirect_to @student_learning_outcome, notice: 'Student learning outcome is successfully created.' }
      format.json { render :show, status: :created, location: 

    else
      format.html { render :new, status: :unprocessable_entity }
      format.json { render :show, status: :unprocessable_entity }
    end
  end
end

```ruby
# LoadStudentQuizResult.start_quiz_upload(params[:file],
10378, "CS", 3100, "Cheatham", "Michelle", "Spring", 2016, 2)
# LoadStudentQuizResult.start_quiz_upload(params[:file],
21334, "CS", 3180, "Buck", "Erik", "Spring", 2016, 1)
# LoadStudentQuizResult.start_quiz_upload(params[:file],
```
@student_learning_outcome
else
format.html { render :new }
format.json { render json: @student_learning_outcome.errors, status: :unprocessable_entity }
end
end
end

# PATCH/PUT
@student_learning_outcomes/1
# PATCH/PUT
@student_learning_outcomes/1.json
def update
respond_to do |format|
if @student_learning_outcome.update(student_learning_outcome_params)
format.html { redirect_to @student_learning_outcome, notice: 'Student learning outcome is successfully updated.' }
format.json { render :show, status: :ok, location: @student_learning_outcome } 47
else
format.html { render :edit }
format.json { render json: @student_learning_outcome.errors, status: :unprocessable_entity }
end
end
end

# DELETE
@student_learning_outcomes/1.json
def destroy
@student_learning_outcome.destroy
respond_to do |format|
format.html { redirect_to student_learning_outcomes_url, notice: 'Student learning outcome is successfully destroyed.' }
format.json { head :no_content }
end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_student_learning_outcome
@student_learning_outcome = StudentLearningOutcome.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def student_learning_outcome_params
end

318
class LoadStudentLearningOutcomesController < ApplicationController
  def index
    LoadStudentLearningOutcome.import_slo_data(params[:file])
    # This is me being lazy to load question titles
    LoadStudentLearningOutcome.import_title_data(params[:file])
    render 'welcome/index'
  end
end

class ProgramEducationalObjectivesController < ApplicationController
  before_action :set_program_educational_objective, only: [:show, :edit, :update, :destroy]

  # GET /program_educational_objectives
  # GET /program_educational_objectives.json
  def index
    @program_educational_objectives = ProgramEducationalObjective.all
  end

  # GET /program_educational_objectives/1
  # GET /program_educational_objectives/1.json
  def show
  end

  # GET /program_educational_objectives/new
  def new
    @program_educational_objective = ProgramEducationalObjective.new
  end

  # GET /program_educational_objectives/1/edit
  def edit
  end

  # POST /program_educational_objectives
  # POST /program_educational_objectives.json
  def create
    @program_educational_objective = ProgramEducationalObjective.new(program_educational_objective_params)
    respond_to do |format|
      if @program_educational_objective.save
        format.html { redirect_to @program_educational_objective, notice: 'Program educational objective is successfully created.' }
        format.json { render :show, status: :created, location: @program_educational_objective }
      else
        format.html { render :new }
        format.json { render JSON爬虫: :unprocessable_entity }  
    end
  end
end
format.json { render json: @program_educational_objective.errors, status: :unprocessable_entity } end end end

# PATCH/PUT /program_educational_objectives/1
# PATCH/PUT /program_educational_objectives/1.json
def update
  respond_to do |format|
    if @program_educational_objective.update(program_educational_objective_params)
      format.html { redirect_to @program_educational_objective, notice: 'Program educational objective is successfully updated.' }
      format.json { render :show, status: :ok, location: @program_educational_objective }
    else
      format.html { render :edit }
      format.json { render json: @program_educational_objective.errors, status: :unprocessable_entity }
    end
  end
end

private

# Use callbacks to share common setup or constraints between actions.
def set_program_educational_objective
  @program_educational_objective = ProgramEducationalObjective.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def program_educational_objective_params
end
File D:\railsApps\playgroundJan2016\config\routes.rb

1 Rails.application.routes.draw do
2 resources :debug_pages
resources :kts_covered_by_answers
resources :slo_averages
resources :class_averages
get 'standard_reports' => 'standard_reports#index'
get 'standard_reports/quiz_coverage' => 'standard_reports#quiz_coverage'
get 'standard_reports/summary_report' => 'standard_reports#summary_report'
post 'standard_reports/abet_report' => 'standard_reports#abet_report'
get 'standard_reports/quiz_results_by_class' => 'standard_reports#quiz_results_by_class'
get 'standard_reports/pedagogy_comparison' => 'standard_reports#pedagogy_comparison'
get 'standard_reports/single_course_report' => 'standard_reports#single_course_report'
get 'standard_reports/abet_data_collection_summary_course_focus' => 'standard_reports#abet_data_collection_summary_course_focus'
get 'standard_reports/abet_data_collection_summary_slo_focus' => 'standard_reports#abet_data_collection_summary_slo_focus'
16 get 'standard_reports' => 'standard_reports#index'
get 'load_student_quiz_results' => 'load_student_quiz_results#index'
post 'load_slo_covered_by_kts/handle_form' => 'load_slo_covered_by_kts#handle_form'
post '/load_internal_reports/load_report' => 'load_internal_reports#load_report'
get 'load_student_learning_outcomes/index'
get 'load_student_learning_outcomes/handle_form'
post 'load_student_learning_outcomes/handle_form' => 'load_student_learning_outcomes#handle_form'
get 'load_student_learning_outcomes/index'
get 'load_internal_reports' => 'load_internal_reports#index'
post '/load_internal_reports/load_report' => 'load_internal_reports#load_report'
get 'cleanup' => 'cleanups#index'
post
'load_student_quiz_results/load_quiz'
=>
'load_student_quiz_results#load_quiz'
resources :slo_covered_by_kts
resources :peo_covered_by_slos
resources :prerequisites
resources :program_of_studies
resources :answers
resources :enrolls
resources :students
resources :student_learning_outcomes
resources :program_educational_objectives
resources :knowledge_topics
get 'welcome/index'
root 'welcome#index'
41
42
43 end
44
File
D:\railsApps\playgroundJan2016\config\secrets.yml
1  # Be sure to restart your server when you modify this file. 2
3  # Your secret key is used for verifying the integrity of signed cookies. 4 # If you change this key, all old signed cookies will become invalid!
5
6  # Make sure the secret is at least 30 characters and all random, 7 # no regular words or you'll be exposed to dictionary attacks.
8  # You can use `rake secret` to generate a secure secret key. 9
10  # Make sure the secrets in this file are kept private 11
# if you're sharing your code publicly. 12
development:
  secret_key_base:
c5dae09ff756add1cb242fdce0fd556
  713fb0304ada7d976aad74da7f8a4c4
  38e2e5ae8c11910f472641071dd9b1
dc54181
  bbdfcb0bfd493acc7b2e83139
15
test:
  secret_key_base:
d16b454e2e99e8d8e776aa7ebe3d9a8
d2564ebf153f20710437a40147712ed
  442cd2cf52dd15508d2f6a5028e9bba
  4073
  4ed7b8f09530ecdl6fbb4b4ef1edee
18
19  # Do not keep production secrets in the repository, 20 # instead read values from the environment.
production:
  secret_key_base: <%= ENV['SECRET_KEY_BASE'] %> 23
File - D:\railsApps\playgroundJan2016\config\database.yml 1 # MySQL. Versions 5.0+ are recommended.
#
# Install the MYSQL driver
# gem install mysql2
#
# Ensure the MySQL gem is defined in your Gemfile
# gem 'mysql2'
#
# And be sure to use new-style password hashing:
#
http://dev.mysql.com/doc/refman/5.0/en/old-client.html
#
default: &default
  adapter: mysql2
  encoding: utf8
  pool: 5
  username: root
  password: root
  host: localhost
  socket: MySQL

development:
  <<: *default
database:
  playgroundJan2016_development

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or
# production.
test:
  <<: *default
database:
  playgroundJan2016_test

# As with config/secrets.yml, you never want to store sensitive
# information, 33 like your database password, in your
# source code. If your source code is 34 ever seen by
# anyone, they now have access to
# your database.
#
# Instead, provide the password as a unix environment variable
# when you boot
# the app. Read
# http://guides.rubyonrails.org/configuring.html#configuring-a-database
# for a full rundown on how to provide these environment variables in
# a production deployment.
#
# On Heroku and other platform providers, you may have a full
# connection URL as an environment variable. For example:
#
DATABASE_URL="mysql2://myuser:mypass@localhost/somedatabase"
#
# You can use this database configuration with:
#
production:
  url: <%= ENV['DATABASE_URL'] %>
password: <%= ENV['PLAYGROUNDJAN2016_DATABASE_PASSWORD'] %>
Appendix C – Retrospective Survey

Findings

Retrospective Survey Findings

By Dr. Clark Shingledecker

The data discussed in this document are obtained using a student survey that formed part of the evaluation process for a revised active learning, inverted classroom approach to teaching an introductory computer science course. The primary objective outcome data being used to assess the effectiveness of the new course include passing grade rates and subsequent content retention tests. The data presented here are derived from a set of survey questions posed to the students after the completion of the course. The primary purpose of 42 of the questions is to explore the relative impact of several key features of the revised pedagogical approach on a set of underlying factors that had been proposed as presenting potential barriers to student success. A second set of four questions included in the survey is designed to record student perceptions of the impact of the overall course on their appreciation of cultural diversity, knowledge of the work conducted and careers available in the computer science and computer
engineering fields, and their degree of identification with the computing professions.

**Impact of Course Elements on Barriers to Success**

129 students in four class sections answered 42 survey questions about the effects of each of the six course elements six features of the revised course (In-Class Activity Focus, Frequent Quizzing, Open-Ended Projects, Problem Solving Exercises, Video Lectures viewed outside of class, and Group-Centered Class Work) on each of five student success factors:

- Interest in Computer Science and Computer Engineering (CS/CE)
- Opportunity (Beliefs about own abilities and capacities) - 2 items
- Psychosocial Factors (Comfort with instructor and class participation, sense of engagement with CS/CE work) – 2 items
- Academic Skills and Knowledge
- Cognitive Skills (Critical thinking and communication)

The students rated the impact of each course element/feature on items describing these success factors/barriers using a five-point scale

1 - Greatly decreased  
2 - Decreased  
3 - No Impact  
4 - Increased  
5 – Greatly increased

**Results**

Overall examination of the data showed that ratings indicating that a particular course element had produced a negative impact (greatly decreased or decreased) on any student success factor are uncommon (Mean 4.2 %). The course element/success factor combinations receiving the highest percentage of negative impact ratings included the “Open-ended projects” and “Group-centered work” on Interest in CS/CE (11% and 9%, respectively), and “Frequent Quizzing” on Confidence in ability to learn CS/CE content (Opportunity).
Because of the low rate of negative impact findings, this analysis focuses on the data reflecting the number of students that reported a positive effect of the course elements (combined ratings of Increased and Greatly Increased) on the success factors. Table 1 shows the percent positive impact (increase) ratings on the success factors for each key element of the revised course format.

<table>
<thead>
<tr>
<th>Key Elements of the Revised Course</th>
<th>INTEREST (My interest in the course)</th>
<th>OPPORTUNITY (My ability to learn the course content)</th>
<th>OPPORTUNITY (My belief in my ability to succeed in the course)</th>
<th>PSYCHOSOCIAL (My comfort with the interactions and class participation)</th>
<th>PSYCHOSOCIAL (Teaching the instructor feels the course content)</th>
<th>ACADEMIC SKILLS/KNOWLEDGE (My learning and mastery of course content)</th>
<th>COGNITIVE SKILLS (My ability to think critically and communicate concepts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity-focused Class Time</td>
<td>96</td>
<td>82</td>
<td>65</td>
<td>72</td>
<td>57</td>
<td>82</td>
<td>88</td>
</tr>
<tr>
<td>Frequent Quizzing</td>
<td>78</td>
<td>74</td>
<td>59</td>
<td>56</td>
<td>58</td>
<td>82</td>
<td>80</td>
</tr>
<tr>
<td>Open-ended Student Projects</td>
<td>63</td>
<td>62</td>
<td>63</td>
<td>47</td>
<td>65</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>Problem-Solving Exercises</td>
<td>79</td>
<td>81</td>
<td>73</td>
<td>64</td>
<td>71</td>
<td>81</td>
<td>81</td>
</tr>
<tr>
<td>Recorded Lectures viewed outside of class</td>
<td>56</td>
<td>47</td>
<td>34</td>
<td>30</td>
<td>29</td>
<td>45</td>
<td>51</td>
</tr>
<tr>
<td>Group-Centered Work</td>
<td>53</td>
<td>67</td>
<td>64</td>
<td>64</td>
<td>71</td>
<td>74</td>
<td>78</td>
</tr>
</tbody>
</table>

Several aspects of the student’s perceptions of the new course design are apparent in this simplified portrayal of the data. Considering the overall impact of the course elements on the combined success factor items, “Activity-focused Class Time”, and “Problem Solving Exercises” received a mean positive impact assessment of > 75%. “Frequent Quizzing”, “Open-Ended Student Projects” and “Group-Centered Class Work” all had combined positive impact ratings > 62% averaged across success factors. Only “Recorded Lectures viewed outside of class” received an average positive impact rating by fewer than half of the
respondents (39%).

To obtain a clearer picture of how the individual course elements differentially affected the success factors, I assigned ranks to the six course elements based on the percent positive impact scores for each survey item. Using these rankings:

Interest in CS/CE

“Activity-focused Class Time” (1) and “Problem Solving Exercises” (2) had the greatest relative positive impact. “Group-Centered Class Work” (5) and “Recorded Lectures viewed outside of class” had the least positive impact on student interest in CS/CE.

Opportunity

The opportunity items focused on self-imposed limits or affordances to opportunity for success created by student own beliefs in their ability to learn CS/CE content and to pursue a CS/CE career. Combining the two survey opportunity item ranks, “Activity-focused Class Time” and “Problem Solving Exercises” tied as the top course elements that positively impacted this factor, while “Problem Solving Exercises” (5) and “Recorded Lectures viewed outside of class” had the smallest positive impact on the opportunity items.

Psychosocial Factors

The impact on psychosocial factors is tapped by two items that addressed the students’ comfort in interacting with the instructor and participating in class, and their belief that they are engaged in real CS/CE work while participating in the course. Unlike the opportunity items, response to these questions diverged somewhat and are discussed separately.
The students reported that their comfort with instructor and class interactions are most positively impacted by “Activity-focused Class Time” (1), “Problem Solving Exercises” and “Group-Centered Class Work” (tied for 2). “Recorded Lectures viewed outside of class” placed a distant 6th in the ranking for positive impact on this factor. The second psychosocial item which assessed the students’ feeling of involvement in real CS/CE work is most positively impacted by “Problem Solving Exercises” and “Group-Centered Class Work” (tied for 1st). “Frequent Quizzing” (5) and “Recorded Lectures viewed outside of class” (6) are rated as having the smallest impact on this engagement component of the psychosocial factor.

Academic Skills and Knowledge in CS/CE

“Activity-focused Class Time” and “Frequent Quizzing” tied for 1st in their positive influence on the students’ academic skills and knowledge, while “Problem Solving Exercises” came in a close second. Somewhat surprisingly, “Recorded Lectures viewed outside of class” is viewed as having the least positive impact on academic factors.

Cognitive Skills

This factor focused on the students’ ability to think critically about, and communicate clearly in written and oral discourse about subject-related concepts and knowledge. Here again, “Activity-focused Class Time” (1) and “Problem Solving Exercises” (2) had the highest relative positive perceived influence. However, “Frequent Quizzing” and “Group-Centered Class Work” are close competitors for second place on this dimension.
Overall Impact of the Revised Course

Four of the survey questions are included to obtain a retrospective evaluation of the overall course. Each of these items is expressed as a positive statement which students could respond to by indicating their level of agreement. For each statement, the 5-point response scale ranged from 1-Strongly Disagree to 5-Strongly Agree with a midpoint of no opinion (3).

Diversity

One general goal of the revised course is to enhance the students’ appreciation of diversity. Presented with the statement “Because of this class, I can better relate and appreciate people of different backgrounds”, 40% of respondents either agreed or strongly agreed. Sixty percent either had no opinion (50%) or disagreed (10%).

Understanding the Nature of CS/CE Work and Careers

Two of the overall course impact survey items addressed students’ understanding of the CS/CE professional work and of career opportunities. 81% of the students agreed that “Because of this course I have a better understanding of who computer scientists and engineers are and what they do” Six percent disagreed and 13 percent had no opinion. When presented with the statement that “Because of this class I have a better understanding of the career opportunities in CS/CE”, 54% agreed, 30% had no opinion and six percent disagreed.

STEM Identity

A final question in this survey attempted to assess student perceptions of the
degree to which they had developed a professional identity associated with the CS/CE fields. Science, Technology, Engineering and Mathematics (STEM) identity has been posited as a key factor in the future persistence of students and professionals working in diverse science and technology fields. The level of identity is normally expressed by students in terms of an affinity or sense of belonging to a particular STEM field and as the extent to which they feel that they have transitioned (or are transitioning) from the mental status of a student learning old knowledge to that of a productive professional who is capable of creating new knowledge. Although the students polled in this survey are new to studies in the field, we asked them to assess the extent to which they might be developing a CS/CE identity by gauging their agreement with the statement: “Because of this course I feel more like a computer scientist or engineer rather than a computer science or engineering student”

Despite their beginner’s standing in CS/CE academic work, 48% of the students either agreed or strongly agreed with this statement. Thirty-four percent gave a neutral response, and 18% registered some level of disagreement.