A. ACTIONS TAKEN TO IMPROVE STUDENT LEARNING
The 2012-2013 academic year saw a complete redesign of all three computer science & engineering programs due to the university-wide transition to semester-based terms. As no assessment data exists for the new semester-based courses or programs of study, assessment efforts during the 2012-2013 cycle has been largely focused on the development of direct assessment instruments and collection of data for the newly offered programs/courses.

Three primary initiatives have been taken to improve student learning during this cycle:
- Delivery of inverted-lecture core sequence (SCALE-UP Classrooms - 152 RC & 355 RC)
- Development of program educational objectives with program constituents
- Preliminary development of infrastructure for continuous assessment of relevant retained knowledge

B. STUDENT LEARNING OUTCOMES ASSESSED AND EXAMINED

Program Educational Objectives (BSCS)
- EXPERT: Graduates of the Computer Science program are employable as computing professionals and will be recognized by their employers as well-prepared for their career in computing
- AGILE: Graduates understand that education is a lifelong process and are well prepared for continuing studies, including graduate studies.
- ENGAGED: Graduates demonstrate appreciation for the professional, social, ethical and leadership roles of computing professionals.
- FOCUSED: Graduates have a set of software theory and development skills that emphasizes software construction, team-based project management, and experience with contemporary software development tools/paradigms.

Student Learning Outcomes (BSCS)
Students who complete the BS in Computer Science will have:

- An ability to apply knowledge of computing and mathematics appropriate to the discipline
- An ability to analyze a problem, and identify and define the computing requirements
- An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs
- An ability to function effectively on teams to accomplish a common goal
- An understanding of professional, ethical, legal, security and social issues and responsibilities
- An ability to communicate effectively with a range of audiences
- An ability to analyze the local and global impact of computing on individuals, organizations, and society
- Recognition of the need for and an ability to engage in continuing professional development
- An ability to use current techniques, skills, and tools necessary for computing practice
- An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.
- An ability to apply design and development principles in the construction of software systems of varying complexity.
The Department of Computer Science and Engineering had 147 Computer Science full majors for Summer 2012 through Spring 2013. Full majors are students who meet the minimum requirements for admission and have a sophomore, junior, or senior status. During Summer 2012 through Spring 2013, 62 students graduated with a Bachelor of Science in Computer Science.

Once the assessment infrastructure is complete, the computer science department expects to collect and examine assessment data for every program SLOs every term of every year. SLOs are mapped to specific knowledge topics developed in specific core courses. These knowledge topics are assessed in subsequent courses. These direct assessments form the examination basis for program SLOs. The initial version of this data collection for this new process was deployed Fall 2012 (as part of the semester curriculum conversion). Development has continued throughout 2013. SLO coverage should be complete sometime in 2014. First analysis of the initial assessment data is presented in the supporting documents.

C. METHODS FOR COLLECTING DATA
Over the past decade, engineering programs nation-wide have devoted significant effort towards measuring educational objectives in the style recommend by ABET Engineering Curriculum 2000 to monitor and improve program effectiveness. In many cases the collection and interpretation of this data has taken place in a labor intensive ad hoc fashion which limits utility of the collected data to drive curricular or pedagogic improvement. We have deployed a data collection infrastructure designed to measure success in retaining specific knowledge area topics deemed critical by our discipline’s professional societies. Where possible, assessment points are deployed at that start of courses that use knowledge topics developed in prerequisite core courses. This infrastructure allows evaluation of retention of expertise, allows assessment of differences in outcomes between learning pathways, and is less subject to instructor, course format, or other bias. Equally important, this infrastructure requires minimal resources post-deployment yet collects critical program data while also providing immediate feedback to students and course instructors regarding the preparation of students as they enter courses that allows for focused review and reinforcement of knowledge areas not retained due to variation in preparation.

We have worked with our program faculty to produce a mapping Knowledge Topics prerequisite to or developed in each of the core/mandatory courses in our program. Our initial assessment framework is limited to mandatory “core” courses. For each course, the faculty has indicated what knowledge topics are developed or assumed (pre-requisite) in the semester-based course offerings. These Knowledge Topics are mapped to one or more SLOs.

We attempt to take a direct assessment of each knowledge topic is made for every student enrolled in every core course in the program every term. These direct assessments take place not in the course that develops the knowledge topic but, when possible, at the beginning of a subsequent course (or courses) that utilize(s) and build(s) on that knowledge topic.

D. ASSESSMENT MEASURES
For each course, the faculty has indicated what knowledge topics are developed or assumed (prerequisite) in the semester-based course offerings. We perform a direct assessment of each knowledge topic not only in the course that develops that knowledge, but when possible, at the beginning of a subsequent course (or courses) that utilizes and builds on the topic. Summative grading rubrics are, when possible, deployed at the start of the next course in the core course. These assessment points allow better evaluation of the retention of expertise as measured prerequisite knowledge coming into each course. Assessment of prerequisite knowledge also allows assessment of differences among learning pathways, and are less subject to instructor-related or course-related bias.

Indirect assessments are obtained from two formal groups, the department’s external advisory board and the department’s student advisory board. The external advisory board consists of alumni, employers, and other professionals familiar with our discipline, program, and students. The department’s student advisory board represents a sample of our undergraduate students, including freshman through seniors, both honors and non-honors students, both traditional and non-traditional students, and both minority and non-minority students.

E. SIGNIFICANT FINDINGS
Given the paucity of historical data under the new semester-based program (delivered for the first time during the 2012-2013 academic year), no findings based on our assessment data can be considered significant. Some potential concerns have been noted and flagged for long-term observation. Initial findings are provided in supporting materials.

F. DISCUSSION OF RESULTS
The results of assessment are shared each term with the department’s undergraduate curriculum committee. Assessments that lead to program/course modification are shared with the department faculty when such changes are proposed for formal action and faculty vote.

G. ACTIONS PLANNED TO IMPROVE STUDENT LEARNING
Development of the new semester-term based infrastructure for assessing retained relevant knowledge continues. A meeting of the department’s undergraduate curriculum committee is scheduled early in Spring term 2014 to discuss the assessments of knowledge developed in Fall 2013 offerings. No specific data-driven actions are planned as a result of the initial data available as of Fall 2013.

H. SUPPORTING DOCUMENTS (recommended)
- Curriculum Assessment document
- Presentation of initial assessment data (from courses taken Fall 2012 and Spring 2013)
- Excerpts from the minutes of the undergraduate studies committee (UGSC)
Infrastructure for continuous assessment of retained relevant knowledge

Travis Doom, Kathleen Timmerman, Michael Raymer
Wright State University, Dayton, OH 45431-0001
Email: travis.doom@wright.edu

Abstract
Over the past decade, engineering programs nation-wide have devoted significant effort towards measuring educational objectives in the style recommend by ABET Engineering Curriculum 2000 to monitor and improve program effectiveness. In many cases the collection and interpretation of this data has taken place in a labor intensive ad hoc fashion which limits utility of the collected data to drive curricular or pedagogic improvement. Herein, we present a data collection infrastructure designed to measure success in retaining specific knowledge area topics deemed critical by each engineering discipline’s professional society. Where possible, assessment points are deployed at that start of courses that use knowledge topics developed in prerequisite core courses. This infrastructure allows evaluation of retention of expertise, allows assessment of differences in outcomes between learning pathways, and is less subject to instructor, course format, or other bias. Equally important, this infrastructure requires minimal resources post-deployment yet collects critical program data while also providing immediate feedback to students and course instructors regarding the preparation of students as they enter courses that allows for focused review and reinforcement of knowledge areas not retained due to variation in preparation.

Introduction
Institutions of higher learning world-wide have embraced continuous improvement models to measure and increase effectiveness of student learning. This is particularly true in undergraduate engineering programs as the ABET 2000 criteria prompted engineering departments to adopt continuous program outcome assessment to satisfy basic level accreditation criteria. All ABET accredited engineering programs are now expected to have some model for continuous program outcome assessment in place. The key to continuous improvement is an effective assessment program. Without a solid measure of student learning, a cycle of improvement is driven by the variations and vagaries of the data and is less likely to result in meaningful positive change.

Learning is multidimensional and requires multiple methods of collection in order to produce meaningful data. Direct methods of assessment measure student performance against some rubric of success. Indirect methods of assessment more often measure the student’s (or observer’s) perception of attainment. While both methods of assessment have their place, direct measures of assessment have been used for decades to provide a means for quality assurance. Historically, direct examinations such as the ACT and SAT have been used to measure the educational achievement of high-school students applying to college. Similarly, 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.

We propose here an infrastructure to assess 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.

**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 updated standards in curriculum, most recently in the volume Computer Science Curricula 2013 (CS2013) [1]. 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 (Figure 1, below). Each of these KAs is further specified as a set of Knowledge Units (Figure 2, below) each of which specifies a set of Knowledge Topics (Figure 3, below) expected at the time of graduation.

<table>
<thead>
<tr>
<th>AL</th>
<th>Algorithms and Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td>Architecture and Organization</td>
</tr>
<tr>
<td>CN</td>
<td>Computational Science</td>
</tr>
<tr>
<td>DS</td>
<td>Discrete Structures</td>
</tr>
<tr>
<td>GV</td>
<td>Graphics and Visualization</td>
</tr>
<tr>
<td>HC</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>IAS</td>
<td>Information Assurance</td>
</tr>
<tr>
<td>IM</td>
<td>Information Management</td>
</tr>
<tr>
<td>IS</td>
<td>Intelligent Systems</td>
</tr>
<tr>
<td>NC</td>
<td>Networking and Communication</td>
</tr>
<tr>
<td>----</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>OS</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>PD</td>
<td>Parallel and Distributed Computing</td>
</tr>
<tr>
<td>PL</td>
<td>Programming Languages</td>
</tr>
<tr>
<td>SDF</td>
<td>Software Development Fundamentals</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>SF</td>
<td>System Fundamentals</td>
</tr>
<tr>
<td>SP</td>
<td>Social and Professional Practice</td>
</tr>
</tbody>
</table>

Figure 1: CS2013 Knowledge Areas [CS2013]

<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>

Figure 2: Sample Knowledge Units in the Algorithms and Complexity Knowledge Area [CS2013]

**AL/Fundamental Data Structures and Algorithms**
- 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)

Figure 3: Sample Knowledge Topics in the Algorithms and Complexity: Fundamental Data Structures and Algorithms Knowledge Unit [CS2013]

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”.

Proceedings of the 2013 ASEE North-Central Section Conference  
Copyright © 2013, American Society for Engineering Education
CS2013 contains 163 lecture hours of Core Tier-1 material and 142 lecture hours of Core Tier-2 material. For comparison, assume that semester-based course has roughly 45 lecture contact hours. The CS2013 then consists of the equivalent of roughly four 3-credit hour semester courses worth of Tier-1 material and roughly three 3-credit hour semester courses worth of Tier-2 material, spread throughout the entire curriculum.

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.

Continuous periodic direct measurements of retained relevant knowledge
We have worked with our program faculty to produce a mapping of which CS2013 Knowledge Topics are prerequisite to or developed in each of the core/mandatory courses in our computer science program. Our initial assessment framework is limited to mandatory “core” courses. Students will gain additional experience in many core knowledge topics in their elective coursework. However, the topics and amount of coverage will necessarily vary based upon the selected electives. Thus, initial observations are limited only to core/mandatory courses.

For each course, the faculty has indicated what knowledge topics are developed or assumed (prerequisite) in the semester-based course offerings. We propose that an appropriate method to assess relevant retained knowledge is to perform a direct assessment of each knowledge topic not only in the course that develops that knowledge, but when possible, at the beginning of a subsequent course (or courses) that utilizes and builds on the topic.

Summative grading rubrics are, when possible, deployed at the start of the next course in the core course sequence (Figure 4, below). These assessment points allow better evaluation of the retention of expertise as measured prerequisite knowledge coming into each course. Assessment of prerequisite knowledge also allows assessment of differences among learning pathways, and are less subject to instructor- or course- related bias.
Figure 4: Assessment points in the first two years of the core curriculum. Additional assessment points exist in advanced core courses including Software Engineering, Operating Systems, and the Capstone Design Sequence. These assessment points are not illustrated. The assessments point for Capstone Design takes places at the end of that course sequence as there is no subsequent required course in the program.

**Immediate value to participants**

We propose that the assessments be required of all students entering every core course. Furthermore, we propose that the results not affect their course grade. As this effort is not associated with a course grade there is no need to proctor or use valuable classroom time on the assessment. The assessments are simply delivered as on-line standardized quizzes (Figure 5, below). We have found that restricting access to on-line classroom materials until the assessment quiz for the course is completed gives complete class participation in the assessment.

In our experience, students are very open about their level of mastery of concepts assessed in not-for-credit surveys of prerequisite knowledge. The feedback from these assessments is immediately useful to students as it calls up old ideas (helping them to be ready for new related knowledge). This immediate feedback can also reduce anxiety regarding the sufficiency of their mastery of assumed prerequisite knowledge or identify specific areas where they can be coached to better prepare for succeed in a new course. As students find the feedback valuable to them personally, they are more likely to give significant and frank effort in the assessment process.

As a direct assessment of student preparedness, this data should be less biased than indirect assessments that ask students their opinion of their ability. Differences in self-expectation that may exist among students due to experience or demographic are removed. Thus, students/faculty get a more accurate measure of how well each student is prepared.
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.

Equally important, the results of these perquisite surveys can be made immediately available to the faculty teaching the course in which the examination is held. If the faculty member sees weakness in prerequisite knowledge then they are able to act to help address the problem immediately. The assessment can help identify individual students that might require additional help as well as identify potential systemic deficiencies introduced by previous poor instruction, variation in schedule due to weather/emergency, differing pathways for preparation (such as transfer courses), or the like. Based upon assessed performance, the faculty can tailor any necessary review of prerequisite topics appropriately to the needs of each term’s student preparation.

**Assessment overhead and administrative burden**

Ease of assessment delivery allows the potential direct assessment of every student every term in every core course. As these assessments are delivered as on-line standardized examination, they require very little class time or faculty effort to administer. Each knowledge topic is mapped to relevant ABET engineering criteria 2000 CAC/EAC a-k criteria listed in ABET’s Criterion 3: Student Outcomes [2,3]. This allows the data to be used by class or longitudinally by student to assess continuous improvement of the program overall against ABET Engineering criteria in a well-defined and straight-forward manner.

The most significant administrative burden is in the initial development, validation, and continuous improvement of the assessment questions. The initial burden of assessment development requires significant faculty involvement and may require multiple years of effort to construct assessment questions for every core course. The measurements for a knowledge area may be skewed by a set of poor assessment questions, thus continuous improvement of the questions in parallel with the improvement of curriculum remains an ongoing administrative effort.
Conclusion
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
Relevant excerpts from the minutes of the Undergraduate Studies Committee (UGSC) for Computer Science and Engineering (CSE) in the College of Engineering and Computer Science (CECS)

Oct 26 2012
- Initial discussion of 2012-2013 CS Program Semester Curriculum Assessment v1.0

Nov 16 2012
- Review of BSCEG PEOs
- Doom will distribute his current slides with possible items to include in the PEOs. The committee will contribute their suggestions for the upcoming Dept. Advisory Board Meeting.

April 05 2013
- Review of Assessment Data
  - Review of initial Knowledge Topics that have success rates of 30% or lower.
  - Several 'poor' questions identified for correction. Associated data to be removed.
  - Potential issues discussed regarding simple data structures: Stacks/Queues and Linked Lists
  - Data this year is insufficient for action
  - Starting next semester one UGSC meeting each term will be devoted to consideration of this assessment data and determining if action should be considered.

September 11, 2013
- Consideration of Program Educational Objectives for update

September 25, 2013
- Program Educational Objectives
  - PEOs revised based on both outside and internal feedback
  - Reduced to four key points for each program
  - CS program provides more depth with FOCUS on software
  - CEG provides a BROAD range
  - Has been set up so it is easily assessable
  - CEG program will be including operating systems to its program with the changes.
  - Formal Vote for the adoption of the new PEO's
  - Vote to adopt the BSCS PEO – APPROVED UNANIMOUSLY
  - Vote to adopt the BSCEG PEO – APPROVED UNANIMOUSLY

November 6, 2013
- CS BA Remove PHL2230 and reintroduce software Engineering CEG4110 to the BACS
  - With the math already in the program this class is redundant
  - Philosophy may have changed the semesters that the class is offered in
  - 1200 to 2200 is the replacement for the math pathway
  - What are we replacing the 3 credit hours with
  - Program directives of the BA program may effect this
- PEO’s for BACS program
  - Keeping unaccredited at the moment keeps it flexible
  - Where should BA program be
  - Same as both BS programs keep Expert, Agile, Engaged
  - Under Agile removed graduate as they are not ready for our graduate program when they graduate
  - 4th section Diverse/Eclectic was changed to Applied
  - Add in software development/management, data science, cyber systems provide a larger set of skills
  - Flexible with other fields
  - Changed data sources to domains
- BA Computer Science is not just the BS program for those who can’t pass the math requirements!
  - Graduates can apply computing and software development principles to a diverse range of Data sources such as analytics, data science, informatics, management, etc
- Difference between BS and BA: More applied, Able to add a minor, for Research/Graduate school – BS, for breadth and variety – BA

- Take PEOs to external Advisory Board for comment as follows:

- BA CS program educational objectives
  - Expert: Graduates of the Computer Science program are employable as computing professionals and will be recognized by their employers as well-prepared for their career in computing
  - Agile: Graduates understand that education is a lifelong process and are well prepared for continuing studies
  - Engaged: Graduates demonstrate an appreciation for the professional, social, ethical, and leadership roles of computing professionals
  - Applied: Graduates can apply computing and software development principles to a diverse range of domains, such as analytics, data science, informatics, management, etc.

December 4, 2013
Program Evaluation Meeting

- New system to evaluate programs
  - Quiz given to collect data
  - Almost done for every core CS cores
  - Seeing if students are learning what they need

- Committee is now going to evaluate the program

- Only 2 semesters of data

- This is the baseline not a complete picture

- Quiz not enforced as it is not graded
  - Should be taken seriously by students

- Need to make the time and data collected as useful as possible

- Most data is on the CS1160, CS1161 vs CS1180 pathways

- CS1181 Data
  - Students in CS1160 spring 13, CS1161 summer 13
  - Students in CS1180 fall 12, spring, or summer 13
  - Overall students do the same on both pathways
    - Primitive types students who take CS1160/1 do half as well (25%) at students who take CS1180 (48%)
    - Might speak to mathematical preparation or focus
    - Where in discreet math
    - May need to focus more on the math skills
    - 1160/1 generally take CS1200 and 2200 CS1180 take MTH 2530
    - Math levels should be equal for both classes
  - Event driven and reactive programming
    - 1160/1 students do better (75%) than 1180 students (52%)
    - Listener objects and listening libraries understood better by 1160/1
    - 1160/1 had more time in the laboratories
    - Very small sample not large enough to be acted on
  - Test students ability to find the answer as well as solve it
  - Question posed asking what the students did choose for the answers vs just the final percentage
  - In future have a percentage breakdown

- SDF/Fundamental Programming Concepts
  - 1160/1 do better (75%) than 1180 (45%)
  - Not a pattern is interesting
  - Need to watch this
  - Possible issue with student just choosing 1st right answer
  - Need to make student look for right answer not just choose 1st one
  - Next question both sets of students did not do well on so possibly too hard

- SDF/Fundamental Data Structures: Arrays
Any question that required students to trace through code and figure out what it will do they did terrible on.

Students are not good at walking through algorithms

Next questions show a pattern of questions like this that students do not do well on

Need to consider what to do about this

CEG3310 Data

- Students took CS1160 fall, CS1161 spring, CS1181 summer
  - Basic type systems: Compound types building
    - None of the 1160/1 (0% all 4 students) got it right but the 1180 (73%) did well.
    - Question probably not too hard but 1160/1 students did not do well
    - 1160 students don’t cover arrays
      - Students have 1160/1 and 1181 at this point
    - First offering in 1160, 1161, and 1181
      - Take with grain of salt
      - Only 4 students
    - Need to watch if see this next time
      - Means we need to spend more time on arrays
      - 1181 doesn’t spend a lot of time on arrays
      - If don’t get in 1161 then not able to fix problem in 1181
      - Problem could be with the need to trace patterns and the students fear of this
  - Fundamental Programming concepts: conditional and iterative control structures
    - Basic and simple question
    - Not good for 1160/1 (0%) not great for 1180 (58%)
    - A number of students in computer org don’t seem to know how to program
    - Don’t have the ability to put together primitives to accomplish a task in any context
    - All students in this class have either 1160, 1161, and 1181 OR 1180, and 1181 before going to this class
    - Possible reason is the way classes were weighted last year
    - Students were able to keep their grades up by completing projects where they could get help
    - This has been changed this year now exams carry more weight so this may make a difference
  - Algorithms and Design: Iterative and recursive mathematical functions
    - CS1160/1 0% CS1180 61%
    - Question as to how many students put recursion for the answer
    - This speaks to the pathway that was taken to get to 1181
    - Need to watch so students that are put into 1181 get the skills necessary to move on
    - Sample is too small to be acted on but if still happening a year from now will need to do something about it
  - Basic type systems: Reference types
    - In Java/C-like languages, variables that store reference types are:
      - Again 1160/1 pathway small sample (0%) 1180 (42%)
      - Hope that students on both pathways take this seriously
  - Fundamental Data Structures: Strings
    - 44% CS1180 able to answer question 0% 1160/1
    - Need rewrite and fix C-like programming
    - By time get through 1180 wording should not be an issue
  - Fundamental data structures: Simple linked structures
    - Nobody did well on this 1160/1 0% 1180 16%
    - Need to let all instructors that teach data structures know the importance of the quiz being taken
      - Bring this up at the faculty meeting
    - Retention rate used to be 85% failure in 1st 2 years now its approximately 60%.
## STUDENT LEARNING OUTCOMES

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>An ability to apply knowledge of computing and mathematics appropriate to the discipline</td>
</tr>
<tr>
<td>b</td>
<td>An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution</td>
</tr>
<tr>
<td>c</td>
<td>An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs</td>
</tr>
<tr>
<td>I</td>
<td>An ability to use current techniques, skills, and tools necessary for computing practice.</td>
</tr>
<tr>
<td>j</td>
<td>An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.</td>
</tr>
<tr>
<td>k</td>
<td>An ability to apply design and development principles in the construction of software systems of varying complexity.</td>
</tr>
</tbody>
</table>
SLO’S FOR ALL QUESTIONS

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1455</td>
</tr>
<tr>
<td>b</td>
<td>818</td>
</tr>
<tr>
<td>c</td>
<td>770</td>
</tr>
<tr>
<td>l</td>
<td>63</td>
</tr>
<tr>
<td>j</td>
<td>294</td>
</tr>
<tr>
<td>k</td>
<td>450</td>
</tr>
</tbody>
</table>

• If a question test multiple SLO’s it is counted multiple times.
SLO FOR ALL QUESTIONS

![Bar chart showing SLO for all questions with categories a, b, c, l, j, and k. The chart displays the proportion of TRUE and FALSE responses for each category.]
## CS 1181 BY SEMESTER

<table>
<thead>
<tr>
<th></th>
<th>Fall 2012</th>
<th>Spring 2013</th>
<th>Summer 2013</th>
<th>Fall 2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>92</td>
<td>321</td>
<td>52</td>
<td>172</td>
</tr>
<tr>
<td>b</td>
<td>46</td>
<td>161</td>
<td>26</td>
<td>86</td>
</tr>
<tr>
<td>c</td>
<td>46</td>
<td>161</td>
<td>26</td>
<td>86</td>
</tr>
<tr>
<td>j</td>
<td>23</td>
<td>78</td>
<td>13</td>
<td>43</td>
</tr>
<tr>
<td>k</td>
<td>23</td>
<td>78</td>
<td>13</td>
<td>43</td>
</tr>
</tbody>
</table>
CS 1181: THROUGH TIME

Fall 2012
Spring 2013
Summer 2013
Fall 2013
### DATA CS 3100 KNOWLEDGE AREA

<table>
<thead>
<tr>
<th>Knowledge Area</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL (Algorithms &amp; Complexity)</td>
<td>32</td>
</tr>
<tr>
<td>AR (Architecture &amp; Organization)</td>
<td>41</td>
</tr>
<tr>
<td>DS (Discrete Structures)</td>
<td>95</td>
</tr>
<tr>
<td>PL (Programming Languages)</td>
<td>16</td>
</tr>
<tr>
<td>SDF (Software Development Fundamentals)</td>
<td>31</td>
</tr>
</tbody>
</table>
CS 3100 BY KNOWLEDGE AREA

AL
AR
DS
PL
SDF

Wrong
Right
# DATA CS 3100 KNOWLEDGE UNIT

<table>
<thead>
<tr>
<th>KA</th>
<th>Knowledge_Unit</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Basic Analysis</td>
<td>16</td>
</tr>
<tr>
<td>AL</td>
<td>Fundamental Data Structures and Algorithms</td>
<td>16</td>
</tr>
<tr>
<td>AR</td>
<td>Assembly level machine organization</td>
<td>16</td>
</tr>
<tr>
<td>AR</td>
<td>Machine-level representation of data</td>
<td>25</td>
</tr>
<tr>
<td>DS</td>
<td>Basics of Counting</td>
<td>63</td>
</tr>
<tr>
<td>DS</td>
<td>Discrete Probability</td>
<td>16</td>
</tr>
<tr>
<td>DS</td>
<td>Graphs and Trees</td>
<td>16</td>
</tr>
<tr>
<td>PL</td>
<td>Object-Oriented Programming</td>
<td>16</td>
</tr>
<tr>
<td>SDF</td>
<td>Algorithms and Design</td>
<td>15</td>
</tr>
<tr>
<td>SDF</td>
<td>Fundamental Data Structures</td>
<td>16</td>
</tr>
<tr>
<td>Pro</td>
<td>Num</td>
<td>sem</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>CS</td>
<td>1181</td>
<td>Fall</td>
</tr>
<tr>
<td>CEG</td>
<td>3320</td>
<td>Fall</td>
</tr>
<tr>
<td>CS</td>
<td>1181</td>
<td>Spring</td>
</tr>
<tr>
<td>CS</td>
<td>3100</td>
<td>Spring</td>
</tr>
<tr>
<td>CS</td>
<td>3100</td>
<td>Spring</td>
</tr>
<tr>
<td>CEG</td>
<td>3310</td>
<td>Spring</td>
</tr>
<tr>
<td>CEG</td>
<td>3320</td>
<td>Spring</td>
</tr>
<tr>
<td>CS</td>
<td>1181</td>
<td>Summer</td>
</tr>
<tr>
<td>CS</td>
<td>3100</td>
<td>Summer</td>
</tr>
<tr>
<td>CS</td>
<td>1181</td>
<td>Fall</td>
</tr>
<tr>
<td>CS</td>
<td>3100</td>
<td>Fall</td>
</tr>
<tr>
<td>CS</td>
<td>3100</td>
<td>Fall</td>
</tr>
<tr>
<td>CEG</td>
<td>3310</td>
<td>Fall</td>
</tr>
</tbody>
</table>
DATABASE UPDATES

- Primary Keys
- CRN became CRN plus year
- Mapped SLOs to PEO
DISCUSSION

• If student retakes the quiz keep both or update answer
  • Current keep both
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>An ability to apply knowledge of computing and mathematics appropriate to the discipline</td>
</tr>
<tr>
<td>b</td>
<td>An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution</td>
</tr>
<tr>
<td>c</td>
<td>An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs</td>
</tr>
<tr>
<td>d</td>
<td>An ability to function effectively on teams to accomplish a common goal</td>
</tr>
<tr>
<td>e</td>
<td>An understanding of professional, ethical, legal, security and social issues and responsibilities</td>
</tr>
<tr>
<td>f</td>
<td>An ability to communicate effectively with a range of audiences</td>
</tr>
<tr>
<td>g</td>
<td>An ability to analyze the local and global impact of computing on individuals, organizations, and society</td>
</tr>
<tr>
<td>h</td>
<td>Recognition of the need for and an ability to engage in continuing professional development</td>
</tr>
<tr>
<td>i</td>
<td>An ability to use current techniques, skills, and tools necessary for computing practice.</td>
</tr>
<tr>
<td>j</td>
<td>An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.</td>
</tr>
<tr>
<td>k</td>
<td>An ability to apply design and development principles in the construction of software systems of varying complexity.</td>
</tr>
</tbody>
</table>
Section 1: Introduction

The conversion of the CS program from a quarter-term based system to a semester-term based system has introduced both new benefits and potential pitfalls. The Faculty has taken advantage of this transition period to redefine introductory course opportunities, to strengthen design experiences, and to provide additional flexibility to our students. However, the breadth of these changes has the potential to have introduced new problems in the delivery of our curriculum and the development of our students.

The Director of Undergraduate Studies for CSE has collected data from faculty teaching or coordinating the semester-based required/core courses of our Computer Science BS program. For each course, the faculty has indicated what knowledge areas are developed or assumed (prerequisite) in the semester-based course offerings. This report presents this data, as well as an initial analysis of the data which identifies potential areas permitting or requiring curricular reform.

The Computer Science Curricula 2013 (CS2013) [Feb 2012 Strawman Draft] developed by the Joint Task Force on Computing Curricula between the ACM and the IEEE-Computer Society is used in this report as a ‘gold standard’ for knowledge areas/topics in contemporary computing education. While acknowledging that every computing program has differing educational objectives, use of the CS2013 will allow our faculty to see how our program compares to a national model and will allow us to reflect on the need, causes, and appropriateness of any major deviations from this widespread consensus proposed by two of computing’s most influential professional organizations.

The CS2013 Body of Knowledge is organized into a set of Knowledge Areas (KAs) which are each subdivided into topics taught throughout a program’s curriculum. Topics are identified as being “core tier-1”, “core tier-2”, or “elective”. CS2013 claims that “a curriculum should include all topics in the tier-1 core and ensure that all students cover this material”. CS2013 also claims that “a curriculum should include all or almost all topics in the tier-2 core and ensure that all students cover the vast majority of this material”. Finally, CS2013 notes that covering only core topics is insufficient for a complete curriculum, “a curriculum should include significant elective material”.

The conversion of the CS program from a quarter-term based system to a semester-term based system has introduced both new benefits and potential pitfalls. The Faculty has taken advantage of this transition period to redefine introductory course opportunities, to strengthen design experiences, and to provide additional flexibility to our students. However, the breadth of these changes has the potential to have introduced new problems in the delivery of our curriculum and the development of our students.

The Director of Undergraduate Studies for CSE has collected data from faculty teaching or coordinating the semester-based required/core courses of our Computer Science BS program. For each course, the faculty has indicated what knowledge areas are developed or assumed (prerequisite) in the semester-based course offerings. This report presents this data, as well as an initial analysis of the data which identifies potential areas permitting or requiring curricular reform.

The Computer Science Curricula 2013 (CS2013) [Feb 2012 Strawman Draft] developed by the Joint Task Force on Computing Curricula between the ACM and the IEEE-Computer Society is used in this report as a ‘gold standard’ for knowledge areas/topics in contemporary computing education. While acknowledging that every computing program has differing educational objectives, use of the CS2013 will allow our faculty to see how our program compares to a national model and will allow us to reflect on the need, causes, and appropriateness of any major deviations from this widespread consensus proposed by two of computing’s most influential professional organizations.

The CS2013 Body of Knowledge is organized into a set of Knowledge Areas (KAs) which are each subdivided into topics taught throughout a program’s curriculum. Topics are identified as being “core tier-1”, “core tier-2”, or “elective”. CS2013 claims that “a curriculum should include all topics in the tier-1 core and ensure that all students cover this material”. CS2013 also claims that “a curriculum should include all or almost all topics in the tier-2 core and ensure that all students cover the vast majority of this material”. Finally, CS2013 notes that covering only core topics is insufficient for a complete curriculum, “a curriculum should include significant elective material”.

Section 1: Introduction

The conversion of the CS program from a quarter-term based system to a semester-term based system has introduced both new benefits and potential pitfalls. The Faculty has taken advantage of this transition period to redefine introductory course opportunities, to strengthen design experiences, and to provide additional flexibility to our students. However, the breadth of these changes has the potential to have introduced new problems in the delivery of our curriculum and the development of our students.

The Director of Undergraduate Studies for CSE has collected data from faculty teaching or coordinating the semester-based required/core courses of our Computer Science BS program. For each course, the faculty has indicated what knowledge areas are developed or assumed (prerequisite) in the semester-based course offerings. This report presents this data, as well as an initial analysis of the data which identifies potential areas permitting or requiring curricular reform.

The Computer Science Curricula 2013 (CS2013) [Feb 2012 Strawman Draft] developed by the Joint Task Force on Computing Curricula between the ACM and the IEEE-Computer Society is used in this report as a ‘gold standard’ for knowledge areas/topics in contemporary computing education. While acknowledging that every computing program has differing educational objectives, use of the CS2013 will allow our faculty to see how our program compares to a national model and will allow us to reflect on the need, causes, and appropriateness of any major deviations from this widespread consensus proposed by two of computing’s most influential professional organizations.

The CS2013 Body of Knowledge is organized into a set of Knowledge Areas (KAs) which are each subdivided into topics taught throughout a program’s curriculum. Topics are identified as being “core tier-1”, “core tier-2”, or “elective”. CS2013 claims that “a curriculum should include all topics in the tier-1 core and ensure that all students cover this material”. CS2013 also claims that “a curriculum should include all or almost all topics in the tier-2 core and ensure that all students cover the vast majority of this material”. Finally, CS2013 notes that covering only core topics is insufficient for a complete curriculum, “a curriculum should include significant elective material”.

Section 1: Introduction

The conversion of the CS program from a quarter-term based system to a semester-term based system has introduced both new benefits and potential pitfalls. The Faculty has taken advantage of this transition period to redefine introductory course opportunities, to strengthen design experiences, and to provide additional flexibility to our students. However, the breadth of these changes has the potential to have introduced new problems in the delivery of our curriculum and the development of our students.

The Director of Undergraduate Studies for CSE has collected data from faculty teaching or coordinating the semester-based required/core courses of our Computer Science BS program. For each course, the faculty has indicated what knowledge areas are developed or assumed (prerequisite) in the semester-based course offerings. This report presents this data, as well as an initial analysis of the data which identifies potential areas permitting or requiring curricular reform.

The Computer Science Curricula 2013 (CS2013) [Feb 2012 Strawman Draft] developed by the Joint Task Force on Computing Curricula between the ACM and the IEEE-Computer Society is used in this report as a ‘gold standard’ for knowledge areas/topics in contemporary computing education. While acknowledging that every computing program has differing educational objectives, use of the CS2013 will allow our faculty to see how our program compares to a national model and will allow us to reflect on the need, causes, and appropriateness of any major deviations from this widespread consensus proposed by two of computing’s most influential professional organizations.

The CS2013 Body of Knowledge is organized into a set of Knowledge Areas (KAs) which are each subdivided into topics taught throughout a program’s curriculum. Topics are identified as being “core tier-1”, “core tier-2”, or “elective”. CS2013 claims that “a curriculum should include all topics in the tier-1 core and ensure that all students cover this material”. CS2013 also claims that “a curriculum should include all or almost all topics in the tier-2 core and ensure that all students cover the vast majority of this material”. Finally, CS2013 notes that covering only core topics is insufficient for a complete curriculum, “a curriculum should include significant elective material”.
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”.

CS2013 contains 163 lecture hours of Core Tier-1 material and 142 lecture hours of Core Tier-2 material. For comparison, assume that semester-based course has roughly 45 lecture contact hours. The CS2013 then consists of the equivalent of roughly four 3-credit hour semester courses worth of Tier-1 material and roughly three 3-credit hour semester courses worth of Tier-2 material, spread throughout the entire curriculum.

In the future, CECS will use the knowledge area/topic scheme of the CS2013 report as a basis for determining student assessment targets in the ongoing regular assessment of our program. This year, as we are developing the student assessment tools, the focus of this assessment will be on the mapping of our curriculum to the national model with an eye towards understand in what ways and why our curriculum might differ from the consensus norm.

The actions that follow from this data may include remedies suggested by any constituent of the program. Initial observations are brought to the Undergraduate Studies director for fact checking (essentially to determine if there is any error/oversight in the raw data), then brought to the Undergraduate Studies committee for further analysis and discussion. The UGSC shall act on the results of their discussion as appropriate. In most cases, observations leading to significant changes will be brought before the external and student advisory boards for consideration, further developed a subset of faculty, and then brought before the full CSE Faculty for final discussion and action.

This initial analysis is limited to mandatory “core” courses in the CSE curriculum. Students will gain additional experience in many core topics in their elective coursework. However, the topics and amount of coverage will necessarily vary based upon the selected electives. Thus, initial observations are limited only to core/mandatory courses.

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Course Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 1180</td>
<td>Computer Science I</td>
</tr>
<tr>
<td>CS 1161</td>
<td>Intro to Computer Programming</td>
</tr>
<tr>
<td>CS 1181</td>
<td>Computer Science II</td>
</tr>
<tr>
<td>CS 3100</td>
<td>Data Structures and Algorithms</td>
</tr>
<tr>
<td>CS 3180</td>
<td>Comparative Languages</td>
</tr>
<tr>
<td>CS 4000</td>
<td>Social Implications of Computing</td>
</tr>
<tr>
<td>CS 2210</td>
<td>Logic for Computer Scientists</td>
</tr>
<tr>
<td>CS 3200</td>
<td>Theoretical Foundations of computing</td>
</tr>
<tr>
<td>CEG 2350</td>
<td>OS Concepts and Usage</td>
</tr>
<tr>
<td>CEG 3310</td>
<td>Computer Organization</td>
</tr>
<tr>
<td>CEG 4110</td>
<td>Introduction to Software Engineering</td>
</tr>
<tr>
<td>CEG 4350</td>
<td>OS Internals and Design</td>
</tr>
</tbody>
</table>
Table 1: WSU Semester-based CS BS required/core courses considered in this analysis

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Course Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH 2530</td>
<td>Elementary Linear Algebra</td>
</tr>
<tr>
<td>ISE2211</td>
<td>Statistics for Engineers</td>
</tr>
<tr>
<td>MTH 2300</td>
<td>Calculus</td>
</tr>
<tr>
<td>MTH 2310</td>
<td>Calculus II</td>
</tr>
<tr>
<td>XXX</td>
<td>Natural Science Lab Sequence</td>
</tr>
</tbody>
</table>

Table 2: WSU Semester-based CS BS required/core courses not considered in this analysis

- CEG 4980/1 Team Projects
- CS2200 Discrete Structures and their Algorithms
- MTH 2570 Discrete Math for Computing
- ENG 1100 English Composition
- EGR 3350 Technical Communication for Engineers and Scientists
Section 2: Core tier-1 topic potential growth areas

Core tier-1 areas are considered 100% mandatory by CS2013. To best guarantee that all students are exposed to 100% of these topics, each of these topics should (ideally) be covered in a mandatory course in the curriculum. Current data suggests that we cover 132 of 163 lecture hours (~80%) of core tier-1 topics in our core curriculum. Core tier-1 KA/topics that have not been identified as being covered in one of our core courses are listed below:

AL Algorithms and Complexity [approximate delta in lecture hours: 2]
- AL/Algorithmic Strategies: Dynamic Programming
- AL/Basic Automata Computability and Complexity: The halting problem

AR Architecture and Organization [100% coverage]

CN Computational Science [approximate delta in lecture hours: 1]
- CN/Fundamentals: Simulation techniques and tools, such as physical simulations, human-in-the-loop guided simulations, and virtual reality
- CN/Fundamentals: Foundational approaches to validating models

DS Discrete Structures [approximate delta in lecture hours: 1]
- DS/Basics of Counting: Arithmetic and geometric progressions
- DS/Basics of Counting: Solving recurrence relations (cross-reference: AL/Basic Analysis)

GV Graphics and Visualization [approximate delta in lecture hours: 2]
- GV/Fundamental Concepts: Basics of Human visual perception (HCI Foundations)
- GV/Fundamental Concepts: Image representations, vector vs. raster, color models, meshes.
- GV/Fundamental Concepts: Forward and backward rendering (i.e., ray-casting and rasterization).
- GV/Fundamental Concepts: Applications of computer graphics: including game engines, cad, visualization, virtual reality.

HC Human Computer Interaction [approximate delta in lecture hours: 3]
- HC/Foundations: Contexts for HCI (anything with a user interface: webpage, business applications, mobile applications, games, etc.)
- HC/Foundations: Processes for user-centered development: early focus on users, empirical testing, iterative design.
- HC/Foundations: Different measures for evaluation: utility, efficiency, learnability, user satisfaction.
- HC/Foundations: Physical capabilities that inform interaction design: color perception, ergonomics
- HC/Foundations: Cognitive models that inform interaction design: attention, perception and recognition, movement, and memory. Gulfs of expectation and execution.
- HC/Foundations: Social models that inform interaction design: culture, communication, networks and organizations.
- HC/Foundations: Principles of good design and good designers; engineering tradeoffs
IAS Information Assurance and Security [approximate delta in lecture hours: 0.5]
- IAS/Fundamental Concepts: Information Assurance Concepts that are key to building an understanding of the IA area.
- IAS/Network Security: TLS
- IAS/Network Security: Hybrid [Public/Secret-key algorithms]

IM Information Management [approximate delta in lecture hours: 0.66]
- IM/Information Management Concepts: Basic information storage and retrieval (IS&R) concepts
- IM/Information Management Concepts: Information capture and representation

IS Intelligent Systems [No mandatory coverage]

NC Networking and Communication [approximate delta in lecture hours: 3]
- NC/Introduction: Organization of the Internet (Internet Service Providers, Content Providers, etc.)
- NC/Introduction: Switching techniques (Circuit, packet, etc.)
- NC/Introduction: Physical pieces of a network (hosts, routers, switches, ISPs, wireless, LAN, access point, firewall
- NC/Introduction: Layering principles (encapsulation, multiplexing)
- NC/Introduction: Roles of the different layers (application, transport, network, datalink, physical)
- NC/Networked Applications: Naming and address schemes (DNS, IP addresses, Uniform Resource Identifiers, etc.)
- NC/Networked Applications: Distributed applications (client/server, peer-to-peer, cloud, etc.)
- NC/Networked Applications: HTTP as an application layer protocol
- NC/Networked Applications: Multiplexing with TCP and UDP
- NC/Networked Applications: Socket APIs

OS Operating Systems [approximate delta in lecture hours: 1]
- OS/Overview of Operating Systems: Mechanisms to support client-server models, hand-held devices
- OS/Operating System Principles: Concepts of application program interfaces (APIs)
- OS/Operating System Principles: Application needs and the evolution of hardware/software techniques

PD Parallel and Distributed Computing [approximate delta in lecture hours: 2.5]
- PD/Parallelism Fundamentals: Programming constructs for creating parallelism, communicating, and coordinating
- PD/Parallelism Fundamentals: Higher-level races (interleavings violating program intention)
- PD/Parallel Decomposition: Independence and partitioning
- PD/Communication and Coordination: Sequential consistency, and its role in programming language guarantees for data-race-free programs
- PD/Parallel Architecture: Multicore processors
- PD/Parallel Architecture: Shared vs. distributed memory
PL Programming Languages [approximate delta in lecture hours: 0.4]
- PL/Functional Programming: Associated language constructs such as discriminated unions and pattern-matching over them

SDF Software Development Fundamentals [100% coverage]

SE Software Engineering [100% coverage]

SF Systems Fundamentals [approximate delta in lecture hours: 11]
- SF/Computational Paradigms: Simple application-level parallel processing: request level (web services/client-server/distributed), single thread per server, multiple threads with multiple servers
- SF/Computational Paradigms: Basic concept of pipelining, overlapped processing stages
- SF/Computational Paradigms: Basic concept of scaling: going faster vs. handling larger problems
- SF/Cross-Layer Communications: Distinction between application and OS services, remote procedure call
- SF/Cross-Layer Communications: Interactions between applications and virtual machines
- SF/Cross-Layer Communications: Reliability
- SF/State State Transition-State Machines: Computers and Network Protocols as examples of State Machines
- SF/System Support for Parallelism: Execution and runtime models that distinguish Sequential vs. Parallel processing
- SF/System Support for Parallelism: System organizations that support Request and Task parallelism and other parallel processing paradigms, such as Client-Server/Web Services, Thread parallelism(Fork-Join), and Pipelining
- SF/System Support for Parallelism: Multicore architectures and hardware support for parallelism
- SF/Performance: Figures of performance merit (e.g., speed of execution, energy consumption, bandwidth vs. latency, resource cost)
- SF/Performance: Benchmarks (e.g., SPEC) and measurement methods
- SF/Performance: CPI equation (Execution time = # of instructions * cycles/instruction * time/cycle) as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations.
- SF/Performance: Amdahl's Law: the part of the computation that cannot be sped up limits the effect of the parts that can

Social and Professional Practice [approximate delta in lecture hours: 2.5]
- SP/Analytical Tools: Ethical argumentation
- SP/Analytical Tools: Ethical theories and decision-making
- SP/ Privacy and Civil Liberties: Ramifications of differential privacy
- SP/ Privacy and Civil Liberties: Technology-based solutions for privacy protection
- SP/ Sustainability: Being a sustainable practitioner, e.g., consideration of impacts of issues, such as power consumption and resource consumption
Section 3: Core tier-2 topic potential growth areas

Initial data suggests that we cover 75 out of 142 lecture hours (approximately 53%) of core tier-2 topics in the mandatory curriculum. Note that some additional coverage of tier-2 material is virtually guaranteed in any elective course. 80% coverage by time of graduation is considered a ‘minimum’ goal by CS2013.

AL Algorithms and Complexity [approximate delta in lecture hours: 4]
- AL/Basis Analysis: Recurrence relations and analysis of recursive algorithms
- AL/Basis Analysis: Some version of a Master Theorem
- AL/Algorithmic Strategies: Branch-and-bound
- AL/Algorithmic Strategies: Heuristics
- AL/Algorithmic Strategies: Reduction transform-and-conquer
- AL/Basic Automata Computability and Complexity: P vs NP (tractable and intractable problems)
- AL/Basic Automata Computability and Complexity: Definition of P, NP, and NP-complete
- AL/Basic Automata Computability and Complexity: Exemplary NP-complete problems (e.g., SAT, Knapsack)

AR Architecture and Organization [approximate delta in lecture hours: 3.5]
- AR/Digital logic and digital systems: Computer-aided design tools that process hardware and architectural representations
- AR/Digital logic and digital systems: Register transfer notation/Hardware Description Language (Verilog/VHDL)
- AR/Digital logic and digital systems: Physical constraints (gate delays, fan-in, fan-out, energy/power)
- AR/Assembly level machine organization: Multiprocessor cache consistency/Using the memory system for inter-core synchronization/atomic memory operations
- AR/Interfacing and communication: External storage, physical organization, and drives
- AR/Interfacing and communication: Buses: bus protocols, arbitration, direct-memory access (DMA)
- AR/Interfacing and communication: Introduction to networks: networks as another layer of access hierarchy
- AR/Interfacing and communication: Multimedia support

CN Computational Science [no Tier-2 coverage implied]

DS Discrete Structures [approximate delta in lecture hours: 1.5]
- DS/Proof Techniques: Well orderings
- DS/Graphs and Trees: Graph isomorphism

GV Graphics and Visualization [approximate delta in lecture hours: 1]
- GV/Fundamental Concepts: Polygonal representation.
- GV/Fundamental Concepts: Basic radiometry, similar triangles, and projection model.
• GV/Fundamental Concepts: Use of standard graphics APIs (see HCI GUI construction).
• GV/Fundamental Concepts: Compressed image representation and the relationship to information theory.
• GV/Fundamental Concepts: Immediate and retained mode.
• GV/Fundamental Concepts: Double buffering.

HC Human Computer Interaction [approximate delta in lecture hours: 2.5]
• HC/Designing Interaction: Principles of different styles of interface: e.g. command line, graphical tangible.
• HC/Designing Interaction: Basic two-dimensional design fundamentals as applied to the visual interface, including use of grid, typography, color and contrast, scale, ordering and hierarchy.)
• HC/Designing Interaction: Basic statistics and techniques for controlled experimentation (especially in regard to web data)
• HC/Designing Interaction: KLM evaluation
• HC/Designing Interaction: Handling human/system failure
• HC/Designing Interaction: User interface standards

IAS Information Assurance and Security [approximate delta in lecture hours: 3.5]
• IAS/Fundamental Concepts: Information Assurance Concepts that are key to building an understanding of the IA area.
• IAS/Fundamental Concepts: National and Cultural Differences including topics such as HIPAA, Safe Harbor, and data protection laws.
• IAS/Fundamental Concepts: Incident Response.
• IAS/Network Security: Authentication protocols
• IAS/Network Security: Digital signatures
• IAS/Network Security: Message Digest
• IAS/Network Security: Network Auditing

IM Information Management [approximate delta in lecture hours: 8.5]
• IM/Information Management Concepts: Information management applications
• IM/Information Management Concepts: Declarative and navigational queries, use of links
• IM/Information Management Concepts: Analysis and indexing
• IM/Database Systems: Approaches to and evolution of database systems
• IM/Database Systems: Components of database systems
• IM/Database Systems: DBMS functions
• IM/Database Systems: Database architecture and data independence
• IM/Database Systems: Use of a declarative query language
• IM/Database Systems: Systems supporting structured and/or stream content
• IM/Data Modeling: Data modeling
• IM/Data Modeling: Conceptual models (e.g., entity-relationship and UML diagrams)
• IM/Data Modeling: Relational data model
• IM/Data Modeling: Object-oriented model
• IM/Data Modeling: Semi-structured data model (expressed using DTD or XML Schema, for example)

**IS Intelligent Systems [approximate delta in lecture hours: 8]**

- IS/Fundamental Issues: Overview of AI problems, Examples of successful recent AI applications
- IS/Fundamental Issues: What is intelligent behavior?
- IS/Fundamental Issues: The Turing test
- IS/Fundamental Issues: Rational versus non-rational reasoning
- IS/Fundamental Issues: Nature of human reasoning
- IS/Fundamental Issues: Nature of environments
- IS/Fundamental Issues: Fully versus partially observable
- IS/Fundamental Issues: Single versus multi-agent
- IS/Fundamental Issues: Deterministic versus stochastic
- IS/Fundamental Issues: Episodic versus sequential
- IS/Fundamental Issues: Static versus dynamic
- IS/Fundamental Issues: Discrete versus continuous
- IS/Fundamental Issues: Nature of Agents
- IS/Fundamental Issues: Autonomous versus Semi-Autonomous
- IS/Fundamental Issues: Reflexive, Goal-based, and Utility-based
- IS/Fundamental Issues: The importance of perception and environmental interactions
- IS/Basic Search Strategies: Problem spaces (states, goals and operators), problem solving by search
- IS/Basic Search Strategies: Factored representation (factoring state into variables)
- IS/Basic Search Strategies: Uninformed search (breadth-first, depth-first, depth-first with iterative deepening)
- IS/Basic Search Strategies: Heuristics and informed search (hill-climbing, generic best-first, A*)
- IS/Basic Search Strategies: Space and time efficiency of search
- IS/Basic Search Strategies: Two-player games (Introduction to minimax search)
- IS/Basic Search Strategies: Constraint satisfaction (backtracking and local search methods)
- IS/Basic Knowledge Representation and Reasoning: Review of propositional and predicate logic (cross-reference DS/Basic Logic)
- IS/Basic Knowledge Representation and Reasoning: Resolution and theorem proving, unification and lifting (propositional logic only)
- IS/Basic Knowledge Representation and Reasoning: Forward chaining, backward chaining
- IS/Basic Knowledge Representation and Reasoning: Review of probabilistic reasoning, Bayes theorem (cross-reference with DS/Discrete Probability)
- IS/Basic Machine Learning: Definition and examples of machine learning for classification
- IS/Basic Machine Learning: Inductive learning
- IS/Basic Machine Learning: Simple statistical-based learning such as Naive Bayesian Classifier, Decision trees
- IS/Basic Machine Learning: Define overfitting problem
- IS/Basic Machine Learning: Measuring classifier accuracy
NC Networking and Communication [approximate delta in lecture hours: 7]
- NC/Reliable Data Delivery: Error control (retransmission techniques, timers)
- NC/Reliable Data Delivery: Flow control (acknowledgements, sliding window)
- NC/Reliable Data Delivery: Performance issues (pipelining)
- NC/Reliable Data Delivery: TCP
- NC/Routing And Forwarding: Routing versus forwarding
- NC/Routing And Forwarding: Static routing
- NC/Routing And Forwarding: Internet Protocol (IP)
- NC/Routing And Forwarding: Scalability issues (hierarchical addressing)
- NC/Local Area Networks: Multiple Access
- NC/Local Area Networks: Local Area Networks
- NC/Local Area Networks: Ethernet
- NC/Local Area Networks: Switching
- NC/Resource Allocation: Need for resource allocation
- NC/Resource Allocation: Fixed allocation (TDM, FDM, WDM) versus dynamic allocation
- NC/Resource Allocation: End-to-end versus network assisted approaches
- NC/Resource Allocation: Fairness
- NC/Resource Allocation: Principles of congestion control
- NC/Mobility: Principles of cellular networks
- NC/Mobility: 802.11 networks
- NC/Mobility: Issues in supporting mobile nodes (home agents)

OS Operating Systems [approximate delta in lecture hours: 2.5]
- OS/Concurrency: Multiprocessor issues (spin-locks, reentrancy) (cross reference SF/Parallelism)
- OS/Scheduling and Dispatch: Deadlines and real-time issues
- OS/Security and Protection: Security methods and devices
- OS/Security and Protection: Protection, access control, and authentication
- OS/Security and Protection: Backups

PD Parallel and Distributed Computing [approximate delta in lecture hours: 9]
- PD/Parallel Decomposition: Basic knowledge of parallel decomposition concepts (cross-reference SF/System Support for Parallelism)
- PD/Parallel Decomposition: Task-based decomposition
- PD/Parallel Decomposition: Implementation strategies such as threads
- PD/Parallel Decomposition: Data-parallel decomposition
- PD/Parallel Decomposition: Implementation strategies such as SIMD and MapReduce
- PD/Parallel Decomposition: Actors and reactive processes (e.g., request handlers)
- PD/Communication and Coordination: Consistency in shared memory models
- PD/Communication and Coordination: Point-to-point versus multicast (or event-based) messages
- PD/Communication and Coordination: Blocking versus non-blocking styles for sending and receiving messages
• PD/Communication and Coordination: Message buffering (cross-reference PF/Fundamental Data Structures/Queues)
• PD/Communication and Coordination: Specifying and testing atomicity and safety requirements
• PD/Communication and Coordination: Granularity of atomic accesses and updates, and the use of constructs such as critical sections or transactions to describe them
• PD/Communication and Coordination: Composition
• PD/Communication and Coordination: Composing larger granularity atomic actions using synchronization
• PD/Communication and Coordination: Transactions, including optimistic and conservative approaches
• PD/Parallel Algorithms, Analysis, and Programming: Critical paths, work and span, and the relation to Amdahl’s law (cross-reference SF/Performance)
• PD/Parallel Algorithms, Analysis, and Programming: Speed-up and scalability
• PD/Parallel Algorithms, Analysis, and Programming: Naturally (embarrassingly) parallel algorithms
• PD/Parallel Algorithms, Analysis, and Programming: Parallel algorithmic patterns (divide-and-conquer, map and reduce, others)
• PD/Parallel Algorithms, Analysis, and Programming: Specific algorithms (e.g., parallel MergeSort)
• PD/Parallel Architecture: Symmetric multiprocessing (SMP)
• PD/Parallel Architecture: SIMD, vector processing

PL Programming Languages [approximate delta in lecture hours: 0.25]
• PL/Event-Driven and Reactive Programming: Separation of model, view, and controller

SE Software Engineering [approximate delta in lecture hours: 7.5]
• SE/Software Project Management: Risk categories including security, safety, market, financial, technology, people, quality, structure and process
• SE/Software Project Management: Risk tolerance (e.g., risk-adverse, risk-neutral, risk-seeking)
• SE/Software Project Management: Risk planning
• SE/Software Project Management: Risk removal, reduction and control
• SE/Tools and Environments: Requirements analysis and design modeling tools
• SE/Tools and Environments: Testing tools including static and dynamic analysis tools
• SE/Tools and Environments: Programming environments that automate parts of program construction processes (e.g., automated builds)
• SE/Tools and Environments: Tool integration concepts and mechanisms
• SE/Software Verification Validation: Verification and validation concepts
• SE/Software Verification Validation: Inspections, reviews, audits
• SE/Software Verification Validation: Testing types, including human computer interface, usability, reliability, security, conformance to specification
• SE/Software Verification Validation: Testing fundamentals
• SE/Software Verification Validation: Unit, integration, validation, and system testing
• SE/Software Verification Validation: Test plan creation and test case generation
• SE/Software Verification Validation: Black-box and white-box testing techniques
• SE/Software Verification Validation: Defect tracking
• SE/Software Verification Validation: Testing parallel and distributed systems
• SE/Software Evolution: Software development in the context of large, pre-existing code bases
• SE/Software Evolution: Software evolution
• SE/Software Evolution: Characteristics of maintainable software
• SE/Software Evolution: Reengineering systems
• SE/Software Evolution: Software reuse
• SE/Software Reliability: Software reliability engineering concepts
• SE/Software Reliability: Software reliability, system reliability and failure behavior (cross-reference SF9/Reliability Through Redundancy)
• SE/Software Reliability: Fault lifecycle concepts and techniques

SF Systems Fundamentals [approximate delta in lecture hours: 7]
• SF/Resource Allocation & Scheduling: Kinds of resources: processor share, memory, disk, net bandwidth
• SF/Resource Allocation & Scheduling: Kinds of scheduling: first-come, priority
• SF/Resource Allocation & Scheduling: Advantages of fair scheduling, preemptive scheduling
• SF/Proximity: Speed of light and computers (one foot per nanosecond vs. one GHz clocks)
• SF/Virtualization and Isolation: Rationale for protection and predictable performance
• SF/Virtualization and Isolation: Levels of indirection, illustrated by virtual memory for managing physical memory resources
• SF/Virtualization and Isolation: Methods for implementing virtual memory and virtual machines
• SF/Reliability through Redundancy: Distinction between bugs and faults, and how they arise in hardware vs. software
• SF/Reliability through Redundancy: How errors increase the longer the distance between the communicating entities; the end-to-end principle as it applies to systems and networks
• SF/Reliability through Redundancy: Redundancy through check and retry
• SF/Reliability through Redundancy: Redundancy through redundant encoding (error correcting codes, CRC/Cyclic Redundancy Codes, FEC/Forward Error Correction)
• SF/Reliability through Redundancy: Duplication/mirroring/replicas

SP Social and Professional Practice [approximate delta in lecture hours: 2]
• SP/Professional Ethics: Dealing with harassment and discrimination
• SP/Professional Ethics: Forms of professional credentialing
• SP/Professional Ethics: Time to market versus quality professional standards
• SP/ Sustainability: Environmental impacts of design choices in specific areas such as algorithms, operating systems, networks, databases, programming languages, or human-computer interaction (cross-reference: HCI/Embedded and Intelligent Systems/Energy-aware interfaces)
Section 4: Strengthening of prerequisite hand-off

Any course that assumes prerequisite topic knowledge must have, as a prerequisite, one or more courses that develop that topic. The following list indicates areas in which this assumption has potentially been violated. For each of the issues below, it must be determined if (A) there is an error in the collected data, (B) course prerequisites are misaligned, or (C) an existing prerequisite is can better develop the knowledge assumed by one or more of its successors.

1. AL/Basic Analysis: Recurrence relations and analysis of recursive algorithms (p for CS3180 and CEG4110)
2. AL/Algorithmic Strategies: Recursive backtracking (p for CS3100, px in CS3180)
3. AL/Algorithmic Strategies: Dynamic Programming (p for CEG4110)
4. AL/Algorithmic Strategies: Branch-and-bound (p for CEG4110)
5. AL/Algorithmic Strategies: Reduction: transform-and-conquer (p for CEG4110)
6. AL/Basic Automata Computability and Complexity: Finite-state machines (p for CEG 4980/1)
7. AR/Digital logic and digital systems: Computer-aided design tools that process hardware and architectural representations (p for CEG 4110)
8. AR/Interfacing and communication: External storage, physical organization, and drives (p for CEG 4110)
9. DS/Proof Techniques: Structural Induction (p for 3100, px for CS3200)
10. DS/Proof Techniques: Recursive mathematical definitions (p for 3100, px for CS3200)
11. DS/Basics of Counting: Arithmetic and geometric progressions (p for CS3100)
12. DS/Basics of Counting: An example of a simple recurrence relation, such as Fibonacci numbers (covered in CS3200 but not CS2210, p for CS3100)
13. DS/Basics of Counting: Basic modular arithmetic (covered in CS3200 but not CS2210, p for CS3100)
15. GV/Fundamental Concepts: Applications of computer graphics: including game engines, cad, visualization, virtual reality. (p for CEG 4110)
16. GV/Fundamental Concepts: Polygonal representation. (p for CEG 4110)
17. GV/Fundamental Concepts: Use of standard graphics APIs (see HCI GUI construction). (p for CEG 4110)
18. IM/Information Management Concepts: Basic information storage and retrieval (IS&R) concepts (p for CEG 4110)
19. IM/Database Systems: DBMS functions (p for CEG 4110)
20. IM/Data Modeling: Data modeling (p for CEG 4110)
21. IS/Fundamental Issues : Overview of AI problems, Examples of successful recent AI applications (p for CEG 4110)
22. IS/Basic Search Strategies: Problem spaces (states, goals and operators), problem solving by search (p for CEG 4110)
23. IS/Basic Machine Learning: Definition and examples of machine learning for classification (p for CEG 4110)
24. NC/Introduction: Organization of the Internet (Internet Service Providers, Content Providers, etc.) (p for CEG 4110)
25. NC/Networked Applications: Naming and address schemes (DNS, IP addresses, Uniform Resource Identifiers, etc.) (p for CEG 4110)
26. OS/Operating System Principles: Concepts of application program interfaces (APIs) (p for CEG 4110)
27. OS/Operating System Principles: Application needs and the evolution of hardware/software techniques (p for CEG 4110)
28. PD/Parallelism Fundamentals: Programming constructs for creating parallelism, communicating, and coordinating (p for CEG 4110)
29. PD/Parallel Decomposition: Independence and partitioning (p for CEG 4110)
30. PD/Parallel Decomposition: Basic knowledge of parallel decomposition concepts (cross-reference SF/System Support for Parallelism) (p for CEG 4110)
31. PD/Communication and Coordination: Sequential consistency, and its role in programming language guarantees for data-race-free programs (p for CEG 4110)
32. PD/Parallel Algorithms, Analysis, and Programming: Critical paths, work and span, and the relation to Amdahl’s law (cross-reference SF/Performance) (p for CEG 4110)
33. PD/Parallel Algorithms, Analysis, and Programming: Speed-up and scalability (p for CEG 4110)
34. PD/Parallel Algorithms, Analysis, and Programming: Naturally (embarrassingly) parallel algorithms (p for CEG 4110)
35. PD/Parallel Algorithms, Analysis, and Programming: Parallel algorithmic patterns (divide-and-conquer, map and reduce, others) (p for CEG 4110)
36. PD/Parallel Algorithms, Analysis, and Programming: Specific algorithms (e.g., parallel MergeSort) (p for CEG 4110)
37. PL/Functional Programming: Associated language constructs such as discriminated unions and pattern-matching over them (p for CEG 4110)

Section 5: Options in the core program
In the core program, there are three major either/or selections allowed as options between core computing curriculum coursework.

CS1180 Computer Science I or CS1161 Intro to Computer Programming II (and its prerequisite). Data suggests that CS1160 (or equivalent prerequisite) plus CS1161 covers all core topics covered by CS1180. From a curriculum coverage perspective, these two options appear to be equivalent.

CS 2210 Logic of Computer Scientists or CS 3200 Theoretical Foundations of Computing. These courses, intentionally, do not cover the same material. However, CS3200 covers two core tier-1 topics (DS/Basics of Counting/Solving recurrence relations and DS/Basics of Counting: Basic modular arithmetic) that are not covered anywhere else in the core curriculum. Similarly, CS2210 covers a small range of core tier-1 topics in DS/Basic logic (Predicate logic, universal and existential quantification, and limitations of propositional and predicate logic) that are not covered anywhere else in the core curriculum. Students taking the either option will miss core tier-1 topics taught only in the other course.
MTH 2570 Discrete Math for Computing or CS 2200 Discrete Structures and their Algorithms. Core tier-1 topic AL/Basic Automata and Complexity/Finite-state machines is covered only in MTH 2570 Discrete Math for Computing. Discrete Structures covers material in DS/Sets, Relations, and Functions (Relations, equivalence relations, functions, inverses, composition) that do not appear to be covered by any other core/required course in the curriculum. Thus, students taking either option may miss the core tier-1 material taught in the other selection.

Section 6: Coverage of Tier-1 and Tier-2 core KA topics

<table>
<thead>
<tr>
<th>KA</th>
<th>CS2012 Knowledge Area</th>
<th>CS2013 Tier-1</th>
<th>CS2013 Tier-2</th>
<th>WSU Tier-1</th>
<th>WSU Tier-2</th>
<th>Δ Tier-1</th>
<th>Δ Tier-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Algorithms and Complexity</td>
<td>19</td>
<td>9</td>
<td>17</td>
<td>4.75</td>
<td>2</td>
<td>4.25</td>
</tr>
<tr>
<td>AR</td>
<td>Architecture and Organization</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>12.5</td>
<td>0</td>
<td>3.5</td>
</tr>
<tr>
<td>CN</td>
<td>Computational Science</td>
<td>1</td>
<td>0</td>
<td>0.33</td>
<td>0</td>
<td>0.66</td>
<td>0</td>
</tr>
<tr>
<td>DS</td>
<td>Discrete Structures</td>
<td>37</td>
<td>4</td>
<td>36</td>
<td>2.5</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>GV</td>
<td>Graphics and Visualization</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>HC</td>
<td>Human Computer Interaction</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1.5</td>
<td>3</td>
<td>2.5</td>
</tr>
<tr>
<td>IAS</td>
<td>Information Assurance</td>
<td>2</td>
<td>6</td>
<td>1.33</td>
<td>2.5</td>
<td>0.66</td>
<td>3.5</td>
</tr>
<tr>
<td>IM</td>
<td>Information Management</td>
<td>1</td>
<td>9</td>
<td>0.33</td>
<td>0.5</td>
<td>0.66</td>
<td>8.5</td>
</tr>
<tr>
<td>IS</td>
<td>Intelligent Systems</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>NC</td>
<td>Networking and Communication</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0.25</td>
<td>3</td>
<td>6.75</td>
</tr>
<tr>
<td>OS</td>
<td>Operating Systems</td>
<td>4</td>
<td>11</td>
<td>3</td>
<td>8.66</td>
<td>1</td>
<td>2.33</td>
</tr>
<tr>
<td>PD</td>
<td>Parallel and Distributed Computing</td>
<td>5</td>
<td>10</td>
<td>2.5</td>
<td>1</td>
<td>2.5</td>
<td>9</td>
</tr>
<tr>
<td>PL</td>
<td>Programming Languages</td>
<td>8</td>
<td>20</td>
<td>7.66</td>
<td>19.75</td>
<td>0.33</td>
<td>0.25</td>
</tr>
<tr>
<td>SDF</td>
<td>Software Development Fundamentals</td>
<td>42</td>
<td>0</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineering</td>
<td>6</td>
<td>21</td>
<td>6</td>
<td>13.5</td>
<td>0</td>
<td>7.5</td>
</tr>
<tr>
<td>SF</td>
<td>Systems Fundamentals</td>
<td>18</td>
<td>9</td>
<td>7.25</td>
<td>2.25</td>
<td>10.75</td>
<td>6.75</td>
</tr>
<tr>
<td>SP</td>
<td>Social and Professional Practice</td>
<td>11</td>
<td>5</td>
<td>8.33</td>
<td>3.25</td>
<td>2.66</td>
<td>1.75</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>163</td>
<td>142</td>
<td>132</td>
<td>75</td>
<td>30</td>
<td>67</td>
</tr>
</tbody>
</table>

Table 3: Approximate curriculum lecture hours summary by knowledge area

If we choose to adhere to the national model, the WSU BS CS should attempt to satisfy 100% of all core tier-1 topics and somewhat more than half of the tier-2 topics currently not covered by our core. In some cases, topic coverage can be included in existing courses with minor effort and little change in focus. In other cases, non-critical material may have to be reduced in an existing core course to accommodate more critical content. Additionally, we must consider added additional courses to the core: such a course might already exist as an elective, or might require design to meet a specific need. A non-exhaustive list of potential remedies for initial consideration now follows. These remedies are intended to serve as points of initial discussion and should not be necessarily considered to be well-researched, complete, or feasible.

Potential Remedy #1: Require Computer Science General Education. The CS department is currently experimenting with a Computing and Technology course. Depending upon the content of this course, it may be worthy of consideration as necessary for graduation in the BS CS program. This option would slightly reduce flexibility for student general education but would not otherwise impact the existing flexibility of the
existing core CS program. Topics from Social and Professional Practice (Ethical argumentation and decision making, professional ethics in computing, effects of computing on privacy and civil liberties, and sustainability), Computational Science (one lecture hour on simulation and modeling), Human Computer Interaction (Interface, cognitive models, social models, principles of good design, accessibility), Information Assurance (Threats, national and cultural differences including HIPPA and data protection laws, use and role of public/secret key encryption, authentication protocols, digital signatures), Intelligent Systems (Overview of AI, Turing test, nature of agents), and Information Management (Basic storage and retrieval concepts, supporting human needs, searching, browsing, navigating, etc.) might be of particular interest to the general education community.

Potential Remedy #2: Computing Systems. Approximately six weeks of uncovered topics are in the areas of Systems Fundamentals. An additional four weeks of uncovered topics in are in the area of parallel and distributed computing, approximately three weeks of uncovered topics are basic topics in networking and communications, and approximately one week of multiprocessor architecture. This material could form the basis for a core systems course. Such a course would align with the recommendations of the external advisory board that we strengthen our focus in systems and may offer an opportunity to strengthen our cyber-security focus.

Potential Remedy #3: Comparative Languages. Data suggests that there may be significant room to replace content currently covered in comparative languages with more contemporary knowledge area topics. This material could include parallel computing (perhaps including study of concurrent/threaded programming or a programming language such as Go), Human Computer Interaction, or Graphics and Visualization. Although not related to the data under study, previous advisory board comments also encourage the study of more net-centric and job-oriented languages (such as C#, .NET, etc.).

Potential Remedy #4: OS Concepts and Usage. Data suggests that the existing core course CEG 2350 covers relatively few core topic area lecture hours. Additionally, many of the covered topics are duplicated elsewhere. There may be room in this course (or in the courses recovering the same topics) to include topics in networking or systems. OS Usage might also a potential location to cover tier-1 core introductory concepts in Human Computer Interface currently covered nowhere else in the core curriculum.

Potential Remedy #5: Information Assurance and Management. Approximately six weeks of uncovered topics in information assurance and information management could, in theory, be covered in a new course in Information. Such a course might also include social and professional practice issues relevant to the collection, storage, and ethical use of information. The remainder of such a course could be used for many purposes, including a more comprehensive introduction to cybersecurity. If such a course was a required/core course for all graduates, it might well provide a distinguishing characteristic to our program graduates that provides value for regional employment.
Potential Remedy #6: Analysis of Algorithms. The mathematical needs of our graduates may be well served by a mathematically rigorous required core course in algorithms that includes significant grounding in proofs, reductions, analysis, and the mathematical concepts used day-to-day by practicing computer scientists. Such a course could cover all uncovered topics in Algorithms and Complexity, as well as uncovered topics in Discrete Structures, Intelligent Systems, and some topics in parallel/distributed computing.

Potential Remedy #7: Logic or/and Theory. The program allows for the choice of CS2210 Logic for CS or CS 3200 Theoretical Foundations for Computing. Both of these courses develop significantly upon discrete structures/mathematics and thus have a significant amount of core topic overlap with discrete structures and each other. However, each course has a small number of different core tier-1 topics covered only in each course. Furthermore, data suggests that CS2200 Discrete structures covers most the core topics covered by CS3200 except for structural induction, recursive mathematical definitions, and the few core topics covered only be CS3200. There may be an opportunity to re-envision the two courses in this option into a single course that has somewhat less overlap with CS2200 Discrete structures and covers all of the core tier-1 material currently covered by either course that exists nowhere else in the curriculum.

Potential Remedy #8: Discrete structures. Both Discrete Structures and Discrete Mathematics cover at least one core tier-1 topic that is not covered by the other course nor elsewhere in the curriculum. Both courses could be updated to address this coverage.

Potential Remedy #9: Calculus II & Matrix Algebra. Topics in Matrix Algebra and Calculus II appear to no longer be considered core tier-1 or tier-2 material by CS2013. These branches of mathematics are of great value in many contemporary areas of elective computing study. Paths exist in our current curriculum which will allow students to have no opportunity to use or further develop these skills beyond the fundamentals introduced in their respective courses. There may be an opportunity to use the mathematics hours currently assigned to Calculus II and/or Matrix Algebra to more thoroughly cover mathematical content in the CS1013 core KAs.

Section 7: Other potential issues observed in the data
Potential Issue #1: Software Engineering: As one of the most advanced courses in the mandatory curriculum, Software Engineering logically assumes the most prerequisite knowledge in incoming students. CEG 4110 (Intro to software engineering) currently has as its only direct prerequisite CS 3100 (Data structures). However, data indicates that it utilizes, as prerequisite knowledge, material from ISE 2111 (Statistics), CEG 4350 (OS), CS3180 (Comparative Languages), and CS 4000 (Social Implications). These linkages need to be addressed. Furthermore, the issue of whether Software Engineering is a pre-/co-
non-requisite for senior project must be addressed for the purposes of assessment and assumed knowledge for capstone projects.

Section 8: Action History
2012 October: Version 1.0 of this report released to WSU CSE Undergraduate studies committee.