SUNY Potsdam

Department of Computer Science

Assessment Plan

October, 2016
This document specifies the learning objectives and outcomes for the BS and BA degrees with a major in Computer Science. Objectives are knowledge, skills, and attitudes that students should possess three or four years after completing the major; that is, they are long term goals. Outcomes are knowledge, skills, and attitudes that students are to be measured against during their progression through the major and that students graduating with this major should have achieved.

Assessment for the Computer Science department is organized into learning outcomes documents, assessment rubrics for each course, a mapping of learning objectives onto our required course offerings, and exam questions tied to the rubrics.

The learning outcomes documents follow the template provided by the Provost's office, and consist of the following sections:

- Intended student learning outcomes
- Connection to the University/Departmental Missions
- Links with other programs/departments
- Measurable Criteria and Assessment Methods
- Data Sources/Results & Analysis
- Application of Results/Action Plan for Improvement

The assessment rubrics define the levels of achievement to be measured for areas in which the department teaches. They are:

1. Mathematics
2. Programming
3. Teamwork
4. Operating Systems
5. Data Structures and Algorithms
6. Professional Ethics
7. Software Engineering
8. Communication

The mapping of learning objectives onto our required course offerings records the levels of achievement expected for outcomes from specific courses. The details of this document is still evolving, especially as 50% of our faculty are new hires. Exam questions that relate to the specific competencies are not included here; they will provide the data on which our analysis will depend.
Objective 1
graduates shall have been prepared for professional careers in a variety of roles such as computer programmer, software engineer, software systems designer, software applications developer, technical software project lead, computer systems analyst, computer systems programmer, software applications tester and maintainer.

Objective 2
graduates shall have acquired the knowledge and skills to do advanced studies and research in computer science and related disciplines.

Objective 3
graduates shall have acquired communication skills – including oral, written, and visual – to be effective team-oriented problem solvers and effective communicators with people who are not trained in computer technology.

Objective 4
graduates shall have acquired the knowledge and skills necessary to participate as effective team members or team leaders in the development of large computer and software systems covering a broad range of applications.
Learning Outcomes

All graduating students with a major in CS (CS Concentration) shall demonstrate:

1. knowledge of discrete and continuous mathematics – including elementary probability and statistics – and the ability to apply logic and mathematical proof techniques to computing problems.

2. knowledge of basic theory of computability and complexity of computation.

3. knowledge of and the ability to apply programming fundamentals in at least two programming languages.

4. knowledge of fundamental data structures and algorithms – including analysis of their correctness and complexity – related to various fields of computer science, and the ability to apply this knowledge to problems through the use of appropriate programming languages.

5. knowledge of computer architecture and organization, computer operating systems, and computer networks, and the ability to apply this knowledge to problems through the use of appropriate programming languages.

6. competence and effectiveness in technical oral, written, and visual communication, particularly as they apply to the dissemination of technical information on subjects dealing with computing technology and applications.

7. knowledge of and skill in applying good practices in software engineering.

8. the ability to function effectively in teams to accomplish a common goal.

9. an understanding of professional, ethical, legal, security, and social responsibilities and issues, including an awareness of impact of computing on individuals, organizations and society.

10. a commitment to continuing professional development.
SUNY Potsdam

Student Learning Outcomes Assessment Plan - Computer Science

Intended Student Learning Outcome #1
Students will demonstrate knowledge of discrete and continuous mathematics - including elementary probability and statistics - and the ability to apply logic and mathematical proof techniques to computing problems.

Connection to University/Departmental Mission
The College mission states in part that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is thus an appropriate component of a liberal arts curriculum. The Departmental mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

Links with other programs/departments
Gen Ed Component
- CIS 105, 125, and 201 all meet the FM requirement
- CIS 300, Foundations of Computer Science, is equivalent to Discrete Mathematics as taught in the Math Department

Measurable Criteria and Assessment Method(s)
Measurable outcomes are given in the attached Outcomes documents. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached outcomes map. Rubric for the various outcomes are also attached. We have completed the rubrics for all courses in the department.

Data Sources/Results & Analysis
Data will come from examination questions in the various courses. Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow the analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

Application of Results/Action Plan for Improvement
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
Intended Student Learning Outcome #2
Students will demonstrate knowledge of basic theory of computability and complexity of computation.

Connection to Univ/Dept Mission
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

Links with other programs/departments
Gen Ed Component
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

Related Courses

Measurable Criteria and Assessment Method(s)
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

Data Source/Results & Analysis
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

Application of Results/Action Plan for Improvement
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
SUNY Potsdam

Student Learning Outcomes Assessment Plan – Computer Science

**Intended Student Learning Outcome #3**
The student will demonstrate knowledge of and the ability to apply programming fundamentals in at least two programming languages.

**Connection to Univ/Dept Mission**
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

**Links with other programs/departments**

*Gen Ed Component*
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

*Related Courses*

**Measurable Criteria and Assessment Method(s)**
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

**Data Source/Results & Analysis**
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

**Application of Results/Action Plan for Improvement**
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
**Intended Student Learning Outcome #4**
The student will demonstrate knowledge of fundamental data structures and algorithms – including analysis of their correctness and complexity – related to various fields of computer science, and the ability to apply this knowledge to problems through the use of appropriate programming languages.

**Connection to Univ/Dept Mission**
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

**Links with other programs/departments**
*Gen Ed Component*
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

*Related Courses*

**Measurable Criteria and Assessment Method(s)**
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

**Data Source/Results & Analysis**
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

**Application of Results/Action Plan for Improvement**
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
**Intended Student Learning Outcome #5**
The student will demonstrate knowledge of computer architecture and organization, computer operating systems, and computer networks, and the ability to apply this knowledge to problems through the use of appropriate programming languages.

**Connection to Univ/Dept Mission**
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

**Links with other programs/departments**
**Gen Ed Component**
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

- **Related Courses**

**Measurable Criteria and Assessment Method(s)**
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

**Data Source/Results & Analysis**
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

**Application of Results/Action Plan for Improvement**
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
SUNY Potsdam
Student Learning Outcomes Assessment Plan – Computer Science

Intended Student Learning Outcome #6
The student will demonstrate competence and effectiveness in technical oral, written, and visual communication, particularly as they apply to the dissemination of technical information on subjects dealing with computing technology and applications.

Connection to Univ/Dept Mission
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

Links with other programs/departments
Gen Ed Component
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

- Related Courses

Measurable Criteria and Assessment Method(s)
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

Data Source/Results & Analysis
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

Application of Results/Action Plan for Improvement
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
SUNY Potsdam

Student Learning Outcomes Assessment Plan – Computer Science

Intended Student Learning Outcome #7
The student will demonstrate knowledge of and skill in applying good practices in software engineering.

Connection to Univ/Dept Mission
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

Links with other programs/departments
Gen Ed Component
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

- Related Courses

Measurable Criteria and Assessment Method(s)
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

Data Source/Results & Analysis
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

Application of Results/Action Plan for Improvement
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
Intended Student Learning Outcome #8
The student will demonstrate the ability to function effectively in teams to accomplish a common goal.

Connection to Univ/Dept Mission
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

Links with other programs/departments
Gen Ed Component
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the W1 requirement

Related Courses

Measurable Criteria and Assessment Method(s)
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

Data Source/Results & Analysis
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis.
This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

Application of Results/Action Plan for Improvement
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
**SUNY Potsdam**

**Student Learning Outcomes Assessment Plan – Computer Science**

**Intended Student Learning Outcome #9**
The student will demonstrate an understanding of professional, ethical, legal, security, and social responsibilities and issues, including an awareness of impact of computing on individuals, organizations and society.

**Connection to Univ/Dept Mission**
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

**Links with other programs/departments**

**Gen Ed Component**
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

**Related Courses**

**Measurable Criteria and Assessment Method(s)**
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

**Data Source/Results & Analysis**
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis. This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

**Application of Results/Action Plan for Improvement**
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.
Intended Student Learning Outcome #10
The student will demonstrate a commitment to continuing professional development.

Connection to Univ/Dept Mission
The College mission states in part, that the College is committed to the liberal arts and sciences as an academic foundation for all students. Computer Science is deeply rooted in science and mathematics and is an appropriate component of a liberal arts curriculum. The Department mission addresses the preparation of computing professionals, and the learning objectives are clearly aligned with this mission.

Links with other programs/departments
Gen Ed Component
- CIS 105, 125, and 201 all meet the FM requirement.
- CIS 380 (Professional Practice) meets the SI requirement
- CIS 405 (Software Engineering) meets the WI requirement

Related Courses

Measurable Criteria and Assessment Method(s)
Measurable outcomes are given in the attached Outcomes document. The mapping of outcomes to the courses that are designed to achieve these outcomes is given in the attached Outcomes Map. A sample rubric for the Programming outcomes is given in the attached rubric. Rubrics for the remaining outcomes are still under development.

Data Source/Results & Analysis
Data obtained from applying the rubrics for the specific outcomes will be obtained from each course and saved in a database, on a per-student basis.
This will allow analysis of the levels of achievement toward meeting the objectives by student, by course, by objective, and overall.

Application of Results/Action Plan for Improvement
Upon analysis of the data collected above, we will determine what areas of our courses need further attention toward meeting the stated objectives.

(Additional Intended Student Learning Outcomes can be added if required)
<table>
<thead>
<tr>
<th>Topic</th>
<th>Acceptable</th>
<th>Marginal</th>
<th>Deficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software design patterns</td>
<td>Can discuss the benefits of design patterns and provide some examples of design patterns</td>
<td>Can describe some benefits of using design patterns and provide at least one example of a design pattern</td>
<td>Can give some example of design patterns</td>
</tr>
<tr>
<td>Software design process</td>
<td>Can discuss the importance of good software design process, and the tradeoffs of different design approaches</td>
<td>Can describe aspects of good software design process and list some tradeoffs</td>
<td>Can list a few aspects of good design process</td>
</tr>
<tr>
<td>Software development methodologies</td>
<td>Can compare and contrast formal, object-oriented, and agile software development methodologies</td>
<td>Can describe some aspects of each of the software development methodologies (formal, object-oriented, agile)</td>
<td>Can describe some ideas of software development methodology in general</td>
</tr>
<tr>
<td>Software documentation</td>
<td>Can discuss the role of and execute basic software documentation, including internal documentation, check-in comments, and project reports</td>
<td>Can write basic software documentation and identify its use</td>
<td>Can identify some aspects of software documentation</td>
</tr>
<tr>
<td>Software version control</td>
<td>Can explain the use of a version control system and explain the importance of version control in software development</td>
<td>Can use a version control system and explain some advantages of such a system</td>
<td>Can use a version control system</td>
</tr>
<tr>
<td>Complexity of large software systems</td>
<td>Can discuss the issues involved in designing and implementing large software systems, and strategies for managing complexity</td>
<td>Can describe some of the challenges in developing large, complex software systems</td>
<td>Can list some problems related to large, complex software systems</td>
</tr>
</tbody>
</table>
## Computer Science Department
### Assessment Rubrics
#### Theory of Computability and Complexity of Algorithms

**Fall 2016**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Acceptable</th>
<th>Marginal</th>
<th>Deficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational complexity</td>
<td>Can thoroughly explain the concept of complexity and its importance in algorithm design</td>
<td>Can explain the basic ideas of complexity and provide some examples of algorithms of different complexity</td>
<td>Can perhaps name some examples of algorithm complexity</td>
</tr>
<tr>
<td>Computational complexity of quadratic and log linear algorithms</td>
<td>Can prove lower bounds for quadratic and log linear sorting algorithms</td>
<td>Can provide analysis and elements of a proof of why an algorithm is quadratic or log linear.</td>
<td>Can describe what it means for an algorithm to be quadratic or log linear.</td>
</tr>
<tr>
<td>Chomsky’s language hierarchy</td>
<td>Can discuss the levels of Chomsky’s language hierarchy in the context of computability.</td>
<td>Can describe the levels of Chomsky’s hierarchy.</td>
<td>Can describe what Chomsky’s language hierarchy is.</td>
</tr>
<tr>
<td>Regular languages</td>
<td>Can apply the regular pumping lemma to prove that a language is not regular</td>
<td>Can provide some analysis or a partial proof using the regular pumping lemma</td>
<td>Can perhaps describe what a regular language is and how the regular pumping lemma works</td>
</tr>
<tr>
<td>Context-free languages</td>
<td>Can apply the context-free pumping lemma to prove that a language is not context-free</td>
<td>Can provide some analysis or a partial proof using the context-free pumping lemma</td>
<td>Can perhaps describe what a context-free language is and how the regular pumping lemma works</td>
</tr>
<tr>
<td>Turing machines</td>
<td>Can describe a Turing machine and design Turing machines for various computational tasks</td>
<td>Can describe a Turing machine and design Turing machines for some tasks</td>
<td>Can describe a Turing machine and perhaps design a machine for a simple task</td>
</tr>
</tbody>
</table>
# Communication Grading Rubric

<table>
<thead>
<tr>
<th>Trait</th>
<th>Exceptional</th>
<th>Acceptable</th>
<th>Amateur</th>
<th>Unsatisfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content:</strong></td>
<td>all content logically organized and appropriately interrelated</td>
<td>content generally logical locally, but not overall</td>
<td>some content appears in the wrong place or is un-explicated</td>
<td>content organization haphazard, concepts not introduced or explained</td>
</tr>
<tr>
<td>\textbf{= organization}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>= completeness, correctness</strong></td>
<td>all content correct and complete (no concepts left un-prepared)</td>
<td>content covers necessary information correctly, but is insufficiently explained</td>
<td>some content incorrect due to usage errors; jargon used and not explained</td>
<td>mistakes in content, essential elements not present</td>
</tr>
<tr>
<td><strong>Visual aids:</strong></td>
<td>all slides easily readable, attractive, eye-catching, appropriately telegraphic, and complete in terms of introducing content</td>
<td>as for &quot;exceptional&quot;, but artistically clunky</td>
<td>slides marginally legible, bland, and somewhat wordy</td>
<td>slides difficult to read, contain grammar/spelling errors, overly wordy</td>
</tr>
<tr>
<td>\textbf{= slides}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>= poster</strong></td>
<td>Poster limited to 300 wds or less, sections match those of the accompanying speech, good use of pictures, graphs, and other non-verbal elements, all necessary sections present</td>
<td>Poster text too mall (too much text), visual flow uncertain, little use of graphical elements</td>
<td>Poster too wordy, some sections missing(e.g. abstract, references, acknowledgements)</td>
<td>Poster elements incorrect (title, author, logos), essential sections missing (methodology, results, conclusions), grammar/spelling errors</td>
</tr>
<tr>
<td><strong>= style</strong></td>
<td>confident appearance, appropriate dress and language, good diction and projection, no hesitations or backing-up for corrections</td>
<td>some lack of confidence, some informal language, occassional diction or projection problems, some evidence of lack of practice</td>
<td>nervousness projected, formality of dress or language insufficient, difficulty being understood, some uncertainty in delivery</td>
<td>debilitating nervousness, physical appearance inappropriate, slang or dialect used, poor projection and diction, obvious lack of preparation</td>
</tr>
<tr>
<td><strong>= grammar, usage, spelling, pronunciation etc.</strong></td>
<td>all grammar and usage correct, spelling in visual aids perfect, pronunciation of special and foreign terms correct</td>
<td>most grammar and usage correct, pronunciation of foreign names unsophisticated</td>
<td>noticeable grammar errors and non-standard usage, one or two spelling errors, clunky pronunciation</td>
<td>poor grammar, use of slang and dialect, more than two spelling errors, multiple pronunciation errors</td>
</tr>
</tbody>
</table>
## Computer Programming Grading Rubric

<table>
<thead>
<tr>
<th>Trait</th>
<th>Exceptional</th>
<th>Acceptable</th>
<th>Amateur</th>
<th>Unsatisfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Specifications</strong></td>
<td>The program works and meets all of the specifications.</td>
<td>The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.</td>
<td>The program produces correct results but does not display them correctly.</td>
<td>The program is producing incorrect results.</td>
</tr>
<tr>
<td><strong>Readability</strong></td>
<td>The code is exceptionally well organized and very easy to follow.</td>
<td>The code is fairly easy to read.</td>
<td>The code is readable only by someone who knows what it is supposed to be doing.</td>
<td>The code is poorly organized and very difficult to read.</td>
</tr>
<tr>
<td><strong>Reusability</strong></td>
<td>The code could be reused as a whole or each routine could be reused.</td>
<td>Most of the code could be reused in other programs.</td>
<td>Some parts of the code could be reused in other programs.</td>
<td>The code is not organized for reusability.</td>
</tr>
<tr>
<td><strong>Documentation</strong></td>
<td>The documentation is well written and clearly explains what the code is accomplishing and how.</td>
<td>The documentation consists of embedded comment and some simple header documentation that is somewhat useful in understanding the code.</td>
<td>The documentation is simply comments embedded in the code with some simple header comments separating routines.</td>
<td>The documentation is simply comments embedded in the code and does not help the reader understand the code.</td>
</tr>
<tr>
<td><strong>Delivery</strong></td>
<td>The program was delivered on time.</td>
<td>The program was delivered within a week of the due date.</td>
<td>The code was within 2 weeks of the due date.</td>
<td>The code was more than 2 weeks overdue.</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td>The code is extremely efficient without sacrificing readability and understanding.</td>
<td>The code is fairly efficient without sacrificing readability and understanding.</td>
<td>The code is brute force and unnecessarily long.</td>
<td>The code is huge and appears to be patched together.</td>
</tr>
</tbody>
</table>
# Computer Science Department
## Assessment Rubrics
### Teamwork
#### Spring 2009

<table>
<thead>
<tr>
<th>Topic</th>
<th>Acceptable</th>
<th>Marginal</th>
<th>Deficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attendance</td>
<td>Attends all except for perhaps one or two group meetings - Notifies the group if a meeting must be missed</td>
<td>Attends most group meetings - Does not notify the group about an absence</td>
<td>Misses many meetings and does not inform the group</td>
</tr>
<tr>
<td>Participation</td>
<td>Takes an active role in group discussions taking into account others contributions - Reports regularly on his/her progress in regard to his/her assigned task</td>
<td>Participates in discussions more often than not or participates but does not consider what others are saying - Reports sporadically on his/her progress</td>
<td>Almost never participates in discussions and does not report on his/her progress unless directly asked.</td>
</tr>
<tr>
<td>Team Role</td>
<td>Makes sure he/she has a clear understanding of his/her team role and pursues it - Precisely understands the roles and responsibilities of other team members</td>
<td>Understands his/her role, if not crisply and pursues it - Has an idea about what others in the group are responsible for</td>
<td>Has no understanding of who does what on the team, including him/herself</td>
</tr>
<tr>
<td>Decision Making</td>
<td>Follows procedures for making decisions participating in the process and strictly abides by group decisions</td>
<td>Sometimes does not follow procedures for making decisions - Does participate in the process and mostly abides by the group decisions made</td>
<td>Make his/her own decisions ignoring those made by the group</td>
</tr>
<tr>
<td>Team Member Support</td>
<td>Treats every team member with respect and acknowledges the work of all team members - Is willing to provide help when asked and is also willing to seek assistance from others or ask questions</td>
<td>Generally respects team members but may disregard some - Acknowledge ment of the work of others is serendipitous rather than planned - Usually provides useful help when asked and usually seeks help from others when needed</td>
<td>Is more often competitive and individualistic rather than supportive</td>
</tr>
<tr>
<td>Conflict Resolution</td>
<td>Consistently 'works to resolve conflict through open discussion and compromise'</td>
<td>Generally works to resolve conflict through open discussion and compromise</td>
<td>Ignores or perhaps inflames conflicts and leaves them unresolved</td>
</tr>
<tr>
<td>Topic</td>
<td>Acceptable</td>
<td>Marginal</td>
<td>Deficient</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OS purposes</td>
<td>Can identify the key features of an operating system</td>
<td>Knows that an operating system is a software layer different from application programs</td>
<td>Thinks that an operating system consists of editors, compilers, and other application software</td>
</tr>
<tr>
<td>OS types</td>
<td>Can distinguish between real-time, time-sharing, and batch operating systems</td>
<td>Knows that different operating systems might be appropriate in different application domains</td>
<td>Thinks that an operating system is what appears on a typical desktop/laptop computer system</td>
</tr>
<tr>
<td>Multiprogramming vs. multiprocessing</td>
<td>Can successfully distinguish between these two terms</td>
<td>Confuses (reverses) the definitions of these two terms</td>
<td>Believes that these two terms mean essentially the same thing</td>
</tr>
<tr>
<td>Processes, processors, and programs</td>
<td>Can identify a process as a program that is being run by a processor</td>
<td>Knows that a processor is necessary to run a program but thinks that process and program mean the same thing</td>
<td>Cannot articulate what a process is</td>
</tr>
<tr>
<td></td>
<td>Can identify and explain the following ingredients of a process and their purposes: instruction pointer, environment pointer, and stack pointer</td>
<td>Knows that parts of a process reside in computer memory but not how the parts interact with the processor</td>
<td>Does not understand that the processor and computer memory interact while a process is running</td>
</tr>
<tr>
<td>Threads</td>
<td>Can correctly state that threads share environments but that processes do not</td>
<td>Knows that threads and processes are similar but cannot articulate how they are different</td>
<td>Cannot distinguish the differences between threads and processes</td>
</tr>
<tr>
<td>Interprocess communication (IPC)</td>
<td>Can identify mutual exclusion and process coordination as key problems in IPC and can give examples of each</td>
<td>Knows that some IPC is necessary but cannot distinguish between mutual exclusion and process coordination</td>
<td>Does not recognize IPC as an operating system issue</td>
</tr>
<tr>
<td><strong>Scheduling</strong></td>
<td><strong>Deadlock</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can correctly state the semantics of semaphores and can give examples of how semaphores can be used to implement mutual exclusion and process coordination</td>
<td>Can explain in detail each of the following scheduling algorithms: first-come first-served, shortest job first, round robin, and priority; can identify which algorithms are used in batch systems and which are used in interactive systems; and can explain how each algorithm affects system behavior</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can describe non-preemptive and preemptive scheduling policies and how each may address different OS goals of throughput, CPU utilization, turnaround time, response time, and meeting deadlines</td>
<td>Can give a formal definition of deadlock, can identify conditions on resource allocations that are necessary and sufficient for deadlock, and can give examples of deadlocked and deadlock-free resource allocations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can describe plausible solutions to the dining philosophers problem and the producer-consumer problem using semaphores</td>
<td>Can give a plausible informal description of deadlock but cannot give examples of deadlocked and deadlock-free resource allocations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can describe plausible solutions to the dining philosophers problem and the producer-consumer problem using semaphores</td>
<td>Cannot describe solutions to either the dining philosophers problem or the producer-consumer problem using semaphores</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Given code examples, can explain whether the use of semaphores is to mutual exclusion or process coordination, but cannot correctly state the semantics</td>
<td>Cannot name a particular process scheduling policy or what scheduling has to do with affecting OS goals</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Given code examples, cannot distinguish between the uses of semaphores for mutual exclusion and for process coordination</td>
<td>Cannot successfully describe what is the purpose of scheduling or how the scheduling algorithms work</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cannot describe solutions to either the dining philosophers problem or the producer-consumer problem using semaphores</td>
<td>Cannot describe deadlock convincingly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can describe explicit strategies for recovery from deadlock and for deadlock prevention and avoidance</td>
<td>Can give plausible approaches to deadlock recovery, prevention, and avoidance, but has difficulty distinguishing among these</td>
<td>Cannot identify any realistic approaches to deadlock recovery, prevention, or avoidance</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td><strong>Memory management</strong></td>
<td>Can distinguish clearly between segmented and paged memory models; can identify the major algorithms for memory allocation for each of these models; and can describe in detail external and internal fragmentation</td>
<td>Knows that segmented and paged memory are different but has difficulty articulating the differences; can identify at least one algorithm for memory allocation for each of these models; but has difficulty distinguishing between external and internal fragmentation</td>
<td>Cannot identify segmentation or paging as memory models</td>
</tr>
<tr>
<td><strong>Input/Output</strong></td>
<td>Can correctly characterize the properties of block and character devices and give examples of each; can describe the purpose of a device driver and where it fits in to the structure of the operating system; can identify how a specific device becomes associated with a device identifier in a process; can correctly describe device interrupt handling (the role of the processor, interrupt vectors, and interrupt handlers) and how it relates to I/O; can identify the purpose of buffering as it refer to I/O devices</td>
<td>Knows that an OS is responsible for managing the transfer of information between I/O devices and user processes but is unclear about how a device driver accomplishes this; knows that a process must access a device but is unclear about how this access occurs in a process; knows that devices can cause interrupts but cannot correctly describe the interrupt handling mechanism; knows that buffering is used for device handling but is unclear about its purpose</td>
<td>Cannot distinguish between block and character devices; cannot identify how a process accesses a device; cannot identify external interrupts or buffering as related to device I/O</td>
</tr>
<tr>
<td>Filesystems</td>
<td>Can describe in general terms the structure of at least one filesystem structure commonly in use and can identify the maximum file, directory, and filename sizes in this structure; can describe the implementation details of hard and symbolic links; can describe the structure of a directory tree; can identify the mechanism by which a process can associate a file with a file descriptor; can describe the implementation of caching and how it affects operating system performance and filesystem integrity.</td>
<td>Knows the name of one filesystem structure commonly in use but cannot describe how to determine its maximum file, directory, and filename sizes; knows that hard and symbolic links are different but cannot describe their implementations; knows how to associate a file with a file descriptor in a program but cannot identify how the operating system achieves this association; knows that caching can improve filesystem performance but cannot describe how caching can affect filesystem integrity.</td>
<td>Cannot name a filesystem structure commonly in use; cannot distinguish between hard and symbolic links; cannot identify how opening a file in a program is associated with operating system primitives; cannot describe how caching can improve filesystem performance.</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Multiprocessor and distributed systems</td>
<td>Can describe the key advantages and complexities of multiprocessor and distributed systems compared to uniprocessor systems; can identify and describe at least one multiprocessor shared memory system architecture (such as SIMD or MIMD); can give an example of code that can be parallelized, and the purpose of doing so; can identify the major synchronization mechanisms used in multiprocessor and distributed systems.</td>
<td>Knows that multiprocessor and distributed systems have advantages over uniprocessor systems but cannot describe their complexities; knows the name a multiprocessor shared memory system architecture but cannot correctly describe it; knows the importance of code parallelization but cannot successfully provide an example of code that can be parallelized; knows that synchronization mechanisms in multiprocessor and distributed systems are important but cannot give specific examples of them.</td>
<td>cannot describe advantages of multiprocessor and distributed systems over uniprocessor systems; cannot name a multiprocessor shared memory system architecture; cannot identify the meaning or importance of code parallelization; cannot identify the purpose of synchronization mechanisms in multiprocessor and distributed systems.</td>
</tr>
<tr>
<td>Operating system security</td>
<td>Can identify and describe the key ideas in security: confidentiality, integrity, authenticity, availability, and non-repudiation; can identify most policies and mechanisms commonly in use in operating systems to implement these; can identify and describe threats to operating system security such as trojan horses, viruses, worms, and root kits</td>
<td>Knows the names of most of the key ideas in security but has difficult describing what they mean; Knows that operating system security is important and can identify two or more mechanisms commonly used to implement computer security policy but has difficult describing these mechanisms; Knows that viruses and such are threats to operating system security but cannot describe how they work</td>
<td>Cannot name most of the key ideas on security; cannot identify more than password protection as a mechanism to implement operating system security</td>
</tr>
<tr>
<td>Topic</td>
<td>Acceptable</td>
<td>Marginal</td>
<td>Deficient</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Indexable Data Structure Use</td>
<td>Can construct an indexable data structure (e.g. array or array list), add elements to it, and access its elements to perform any appropriate task</td>
<td>Can construct an indexable data structure, add elements to it, and access its contents in sequence</td>
<td>Can construct an indexable data structure, but cannot add data to it or access its elements</td>
</tr>
<tr>
<td>Linked Lists</td>
<td>Can write an appropriate class to implement linked list nodes and use it to build a linked list of data, traverse the linked list, find and delete data from the linked list, and reverse the linked list</td>
<td>Can write an appropriate class to implement linked list nodes and use it to build a linked list of data, and traverse the linked list</td>
<td>Cannot write an appropriate class to implement linked list nodes, or, cannot use the class to build a linked list of data and traverse it</td>
</tr>
<tr>
<td>Stacks and Queues</td>
<td>Can describe the purpose and use of a stack or a queue, and given an implementation of a stack/queue can use it to solve an appropriate problem - Can extend a stack/queue for use in a different context - Can design and implement a stack/queue class</td>
<td>Can describe the purpose and use of a stack or a queue, and given an implementation of a stack/queue can use it to solve an appropriate problem</td>
<td>Can perhaps describe the purpose and use of a stack or a queue, but cannot use it to solve a problem</td>
</tr>
<tr>
<td>Binary Trees</td>
<td>Can describe the purpose and use of a binary tree, and given an implementation of one, can use it to solve an appropriate problem - Can write an appropriate class to implement binary tree nodes and use it to build a binary tree, traverse the tree using different tree traversals, find and delete data from the binary tree and add data to the binary tree used either as an unordered tree or a binary search tree</td>
<td>Can describe the purpose and use of a binary tree, and given an implementation of one, can use it to solve an appropriate problem - Can write a class to implement binary tree nodes and use it to build a binary tree either as an unordered tree or a binary search tree</td>
<td>Can perhaps describe the purpose and use of a binary tree, and given one, can perhaps use it</td>
</tr>
<tr>
<td>Graphs</td>
<td>Can describe the purpose and use of a graph, and given an implementation of one, can use it to solve an appropriate problem - Can write an appropriate class to implement a graph and use it to traverse the graphs using different graph traversals</td>
<td>Can describe the purpose and use of a graph, and given an implementation of one, can use it to solve an appropriate problem</td>
<td>Can perhaps describe the purpose and use of a graph, and given one, can perhaps use it minimally, but not to solve a problem</td>
</tr>
<tr>
<td>Searching</td>
<td>Can identify and implement a linear search and a binary search and knows when each can be used - Can produce an analysis of why linear search is linear and why binary search is logarithmic</td>
<td>Can identify a linear search and a binary search and knows when each is used - Can implement a linear search but cannot successfully implement a binary search - Can identify the difference between the two in runtime complexity.</td>
<td>Can perhaps identify a linear search from a binary search.</td>
</tr>
<tr>
<td>Quadratic Sorting</td>
<td>Can identify and implement each of selection sort, insertion sort, and bubble sort, and can produce an informal analysis of why each is a quadratic sorting algorithm</td>
<td>Can implement at least one quadratic sorting algorithm and identify two others</td>
<td>Can perhaps identify one quadratic sorting algorithm.</td>
</tr>
<tr>
<td>n log n Sorting</td>
<td>Can identify and implement each of merge sort, quick sort, and heap sort, and can produce an informal analysis of why each sort is $n \log n$.</td>
<td>Can implement at least one $n \log n$ sorting algorithm and identify two others.</td>
<td>Can perhaps identify one $n \log n$ sorting algorithm.</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>Recursive Algorithms</td>
<td>Can describe recursion, and can write a recursive algorithm to implement most operations on stacks, queues, trees, and graphs. Can describe tail recursion.</td>
<td>Can describe recursion, and can write a recursive algorithm to implement several operations on stacks, queues, trees, and graphs.</td>
<td>Can perhaps describe recursion and implement one simple problem recursively (e.g. factorial).</td>
</tr>
</tbody>
</table>
Computer Science Department
Assessment Rubrics:
Professional Ethics

Spring 2009

<table>
<thead>
<tr>
<th>Topic</th>
<th>Acceptable</th>
<th>Marginal</th>
<th>Deficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation of the</td>
<td>Can produce a technical explanation that is both concise and complete and</td>
<td>Can produce a technical explanation but either incomplete or rambling</td>
<td>Can name the technical issue, but the explanation of it is misleading, inac-</td>
</tr>
<tr>
<td>Technical Issue</td>
<td>that introduces the ethical issues</td>
<td></td>
<td>curate, or non existent</td>
</tr>
<tr>
<td>Identification of</td>
<td>Can specify who is impacted by an ethical dilemma and how they are</td>
<td>Can specify who is impacted by an ethical dilemma and how they are</td>
<td>Can specify who is impacted by the ethical dilemma or how they are</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>impacted - Can identify all the important values involved and how they are</td>
<td>impacted - Attempts to explain the values involved but leaves out</td>
<td>impacted, but not both or perhaps neither. Cannot identify the values</td>
</tr>
<tr>
<td></td>
<td>ethically significant</td>
<td>important points</td>
<td>involved.</td>
</tr>
<tr>
<td>Analogy Use</td>
<td>Can make an appropriate analogy to the ethical issue that is particularly</td>
<td>Can make an appropriate analogy to the ethical issue</td>
<td>Can make an analogy that is not well used, incorrect, or cannot make an</td>
</tr>
<tr>
<td></td>
<td>revealing</td>
<td></td>
<td>analogy to the ethical issue</td>
</tr>
<tr>
<td>Evidence</td>
<td>All statements of fact are accurate</td>
<td>Most statements of fact are accurate</td>
<td>Most or all statements of fact are inaccurate</td>
</tr>
<tr>
<td>Support for Opinions</td>
<td>Can provide strong support for opinions</td>
<td>Can provide adequate support for opinions</td>
<td>Can provide no or little support for opinions</td>
</tr>
<tr>
<td>Documentation</td>
<td>Can identify all sources of information and reference them appropriately</td>
<td>Can identify most sources of information and reference them appropriately</td>
<td>Sources of information are not appropriate or not identified</td>
</tr>
<tr>
<td>Counterarguments</td>
<td>Can present counterarguments and analyze them</td>
<td>Can present counterarguments but cannot analyze them</td>
<td>Cannot present a counterargument</td>
</tr>
<tr>
<td>Social/Ethical Analysis</td>
<td>Can use ethical and social theories convincingly</td>
<td>Uses ethical and social theories but not convincingly</td>
<td>Does not make use of ethical and social theories</td>
</tr>
</tbody>
</table>
## Computer Science Department
### Assessment Rubrics
#### Software Engineering Practice

**Fall 2016**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Acceptable</th>
<th>Marginal</th>
<th>Deficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software design patterns</td>
<td>Can discuss the benefits of design patterns and provide some examples of design patterns</td>
<td>Can describe some benefits of using design patterns and provide at least one example of a design pattern</td>
<td>Can give some example of design patterns</td>
</tr>
<tr>
<td>Software design process</td>
<td>Can discuss the importance of good software design process, and the tradeoffs of different design approaches</td>
<td>Can describe aspects of good software design process and list some tradeoffs</td>
<td>Can list a few aspects of good design process</td>
</tr>
<tr>
<td>Software development methodologies</td>
<td>Can compare and contrast formal, object-oriented, and agile software development methodologies</td>
<td>Can describe some aspects of each of the software development methodologies (formal, object-oriented, agile)</td>
<td>Can describe some ideas of software development methodology in general</td>
</tr>
<tr>
<td>Software documentation</td>
<td>Can discuss the role of and execute basic software documentation, including internal documentation, check-in comments, and project reports</td>
<td>Can write basic software documentation and identify its use</td>
<td>Can identify some aspects of software documentation</td>
</tr>
<tr>
<td>Software version control</td>
<td>Can explain the use of a version control system and explain the importance of version control in software development</td>
<td>Can use a version control system and explain some advantages of such a system</td>
<td>Can use a version control system</td>
</tr>
<tr>
<td>Complexity of large software systems</td>
<td>Can discuss the issues involved in designing and implementing large software systems, and strategies for managing complexity</td>
<td>Can describe some of the challenges in developing large, complex software systems</td>
<td>Can list some problems related to large, complex software systems</td>
</tr>
</tbody>
</table>
Student Learning Outcomes
(1) Mathematics

Tuesday 11th October, 2016

Program SLO (1): All graduating students with a major in CS (CS Concentration) shall demonstrate knowledge of discrete and continuous mathematics including elementary probability and statistics and the ability to apply logic and mathematical proof techniques to computing problems.

CIS 125: Probability and Statistics (3 - mastery, C - substantial)

Measurable Outcome 1: Students can calculate simple probability.

Measurable Outcome 2: Students can calculate conditional probability.

Measurable Outcome 3: Students can use simple or conditional probability when appropriate.

Measurable Outcome 4: Students can define and calculate mean, median, and mode of a data set.

Measurable Outcome 5: Students can describe the differences between data sets from differences in summary values.

Measurable Outcome 6: Students can define normal distribution, mean, and standard deviation.

Measurable Outcome 7: Students can apply the statistics of normal distribution given the mean and deviation.

CIS 203: Computer Science II (1 - recognition, A - minimal)

Measurable Outcome 1: Students define big-O for worst case algorithm run-times.

Measurable Outcome 2: Students can sort simple big-O classes in ascending order of run-time.

Measurable Outcome 3: Students can calculate the big-O run-time of sorts, searches, and simple programs with nested loops.
CIS 300: Foundations of Computer Science  (3 - mastery, C - substantial)

Measurable Outcome 1: Students compute truth values, truth tables, and logical equivalence of statements in the Boolean algebra, both quantified and unquantified.

Measurable Outcome 2: Students define proposition and predicate; they can translate English descriptions into propositions and predicates appropriately.

Measurable Outcome 3: Students can apply first-order logical rules of argument.

Measurable Outcome 4: Students can manipulate sets and convert set descriptions to logic about membership and back again.

Measurable Outcome 5: Students can identify direct proof, proof by contradiction, and proof by induction when reading a proof.

Measurable Outcome 6: Students can construct proofs (direct, indirect, inductive) for simple statements in set theory and other fields of mathematics.

Measurable Outcome 7: Students can differentiate between countable and uncountable sets.

Measurable Outcome 8: Students can describe the Halting Problem and the proof that it is not computable.

CIS 301: Theory of Computation  (3 - mastery, B - moderate)

Measurable Outcome 1: Students define language, string, and alphabet. They can relate languages and the computation of simple functions.

Measurable Outcome 2: Students define regular language, deterministic finite automaton, nondeterministic finite automaton, and regular expression.

Measurable Outcome 3: Students can prove a regular language is regular; students can prove a context-free language context-free.

Measurable Outcome 4: Students can apply both pumping lemma to prove given languages are not regular or not context-free.

Measurable Outcome 5: Students define algorithm.

Measurable Outcome 6: Students interpret and design Turing machines for simple algorithms.

Measurable Outcome 7: Students prove the Halting Problem is not computable with a Turing machine.

CIS 303: Algorithm Analysis  (3 - mastery, B - moderate)
Measurable Outcome 1: Students can design efficient programs with mathematical data structures including sets, graphs, and sequences.

Measurable Outcome 2: Students can analyze the run-time complexity of recursive algorithms.

Measurable Outcome 3: Students can discuss the proof that the class of general sorting algorithms have a lower-bound in big-O run-time of $n \cdot \log(n)$.

CIS 356: Assembly Language and Computer Architecture (2 - familiarity, B - moderate)

Measurable Outcome 1: Students translate to/from binary, decimal, and hexadecimal number systems.

Measurable Outcome 2: Students convert/interpret signed integral values in multiple encodings to include 2s-compliment.

Measurable Outcome 3: Students can define "endianess" in multi-byte quantities.

Measurable Outcome 4: Given a set of logic gates, students can prove it is logically complete.

Measurable Outcome 5: Students differentiate between combinatoric and sequential logic circuits.

Measurable Outcome 6: Students design simple logic circuits. Students combine simple logic circuits into an ALU.

CIS 410: Computer Networks (1 - recognition, A - minimal)

Measurable Outcome 1: Students can describe how endianess impacts inter-computer communication.

Measurable Outcome 2: Students can program translation of different multi-byte data encodings for heterogeneous networks.

CIS 420: Database Systems (3 - mastery, B - moderate)

For relational algebra, can substitute relational calculus.

Measurable Outcome 1: Students can apply the relational algebra to creation, retrieval, update, and deletion of data in a database.

Measurable Outcome 2: Students can define database normal forms and translate a database design into at least a third-normal form.

Measurable Outcome 3: Students can explain anomalies caused by unnormalized data and how normalized databases avoid them.
<table>
<thead>
<tr>
<th></th>
<th>CS 201</th>
<th>CS 202</th>
<th>CS 300</th>
<th>CS 301</th>
<th>CS 303</th>
<th>CIS 310</th>
<th>CIS 356</th>
<th>CIS 380</th>
<th>CIS 405</th>
<th>CIS 410</th>
<th>CIS 420</th>
<th>CIS 480/490</th>
<th>MATH 125</th>
<th>MATH 151</th>
<th>MATH 152</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CS I</td>
<td>CS II</td>
<td>Discrete</td>
<td>Theory</td>
<td>Algorithms</td>
<td>OS</td>
<td>Arch.</td>
<td>Ethics</td>
<td>SE</td>
<td>Networks</td>
<td>Databases</td>
<td>Capstone</td>
<td>Prob/Stat</td>
<td>Calc I</td>
<td>Calc II</td>
</tr>
<tr>
<td>Math</td>
<td>1A</td>
<td>3C</td>
<td>3B</td>
<td>3B</td>
<td>2B</td>
<td>1A</td>
<td>3B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2C</td>
<td>3C</td>
<td>3C</td>
</tr>
<tr>
<td>Theory</td>
<td>0</td>
<td>3C</td>
<td>2B</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming</td>
<td>2C</td>
<td>3C</td>
<td>3A</td>
<td>3C</td>
<td>3B</td>
<td>3C</td>
<td>3C</td>
<td>3B</td>
<td>3C</td>
<td>3C</td>
<td>3C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithms</td>
<td>1A</td>
<td>2B</td>
<td>2A</td>
<td>2C</td>
<td>3C</td>
<td>2B</td>
<td>0</td>
<td>2B</td>
<td>2A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arch/OS/Networks</td>
<td>0</td>
<td></td>
<td></td>
<td>3C</td>
<td>3C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Engineering</td>
<td>1A</td>
<td>1A</td>
<td></td>
<td>2A</td>
<td>0</td>
<td>3C</td>
<td>2A</td>
<td>2A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teamwork</td>
<td>2B</td>
<td>2B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional Practice</td>
<td>1A</td>
<td>0</td>
<td></td>
<td></td>
<td>3C</td>
<td>2A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional Dev.</td>
<td>1A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Proficiency
1 = recognition
2 = familiarity
3 = mastery

### Coverage
A = minimal
B = moderate
C = substantial