Table 2. Program-Level Learning Outcomes Developed by CS/SE Faculty at North Carolina State University.

Program-Level Learning Outcomes

To demonstrate that graduates can reason effectively about computing and develop software, they should be able to:

1. Identify and define abstract computing models that could provide a basis for solving a given problem and analyze them for their potential and limitations for a solution.

2. Prove mathematically the characteristics and limitations of an abstract model of computation with respect to the ability to solve specific abstract problems and/or to do so efficiently; inherent in this ability is the mastery of techniques such as (a) decomposing and synthesizing instances, (b) providing the equivalence of different models, (c) searching for patterns in the various instances, (d) proving that certain patterns fit the model and others do not fit the model, and (e) determining the extent to which a model can solve the problem and solve it with acceptable use of resources as defined mathematically.

3. Develop efficient algorithms and data structures for solving a problem and identify other problems or algorithms to which these apply.

4. Recognize and define a problem related to a specific scenario that can be solved with a software application. Describe how the end-users or internal actors within a system intend to use the application to be developed. Gather and analyze information that allows for requirements that will solve the problem to be created; validated; verified; and, if necessary, revised.

5. Create and express a design for an underlying abstract model of computation that accommodates defined system requirements—including considerations of privacy, security, and efficiency—so that a developer can implement the application. Review the design to ensure it can accomplish the requirements and, where it does not, redesign until it meets the requirements.

6. Implement software conforming to a specified design so that it is usable, testable and modifiable by others. Review the implementation to ensure it meets the system requirements and conforms to design and, where it does not, correct the implementation until it meets the requirements and design.

7. Plan and execute appropriate tests in order to identify ways in which the software does not meet the requirements and, where it does not, to redesign, implement and retest until it meets the requirements.

The following **communication outcomes** are derived from the general program outcomes above. By achieving these communication outcomes, students both learn to do what is described in the general outcomes and demonstrate that they have attained those outcomes.

To demonstrate that graduates have achieved the general program learning outcomes, they should be able to:

1. Present in writing or orally an abstract model that could be used to solve a real-world application problem so that the presentation could be understood by stakeholders.

2. Write a mathematical proof related to an abstract model of computation so that it can be understood by an audience with sufficient mathematical maturity (ability to understand proofs by induction, contradiction, etc.)

3. Present in writing or orally the reasoning they have applied in creating a mathematical proof related to an abstract model of computation so that it can be understood by someone acquainted with an application of the model.

4. Present in writing or orally a description of how an abstract model of computation can be productively applied to solving a problem related to software engineering in another area of computer science or in another field

5. Present in writing or orally a critical assessment of a problem situation defined by a need for software to be developed for solving the problem: (a) collect information from sponsors, end-users, and on-site observations; (b) analyze that information; (c) use the analysis to define the problem in terms of the stakeholders' needs and goals for addressing those needs

6. Write requirements representing the stakeholders' needs and goals in such a way that the requirements can be applied in a design by others

7. Read requirements for various purposes, such as to inspect and correct them, to validate them as meeting the user's needs, to revise them so that they better meet user's needs, to implement them in a design, and to identify what students don't know and what they need to know to create code.

8. Write a design that accommodates the defined system requirements—including considerations of privacy, security, and efficiency—so that a developer can implement the application.

9. Read a design for various purposes, such as to ensure it can accomplish the requirements and, where it does not, redesign until it meets the requirements and to translate it into code.

10. Write a program to conform to a specified design so that it is usable, testable, and modifiable by others.

11. Write a narrative description of code, including a list of file names or directories included.

12. Read code and comments for various purposes, to find and correct errors in syntax and semantics, to determine what a program is supposed to do, to revise a program so that it accomplishes what it is supposed to do, to modify a program for different purposes, to ensure that a program conforms to system requirements and conforms to design, to provide productive feedback to those who created it, to continue a program begun by someone else, and to apply it to new uses.

13. Write a developer guide that is appropriate to the audience.

14. Write a user guide that is appropriate to the audience.

15. Present in writing or orally a test plan and results of testing that identifies ways in which the software does not meet the requirements.

16. Present in writing or orally progress reports that describe advancements and difficulties in a software development project.

17. Present in writing and orally a full technical report describing a software development project.

18. Read technical literature in the field for various purposes, such as to summarize, to analyze it, to answer a technical question, and to solve a technical problem.

19. Present in writing or orally a research report that solves a technical problem based on an analysis of literature in the field.

20. Work effectively in teams: (a) develop ground rules to guide the team's approach to work; (b) define roles so that expectations of team members are clear and followed; (c) create agendas and minutes for team meetings; (d) interact with other team members in ways that assure the productive contributions of all team members; (e) create specific action items for each member and then hold him or her accountable; (f) identify, create, and manage the tools that enable teams to work effectively; (g) resolve conflicts among team members.

For the CS2 course, faculty also developed new communication-centric outcomes:

- Interpret a UML diagram and explain its relationship to a problem statement.
- Read and understand code written by people other than themselves.
- Use a problem statement to define a set of software requirements.
- Explain how a final software implementation deviated from their original design.
- Follow good programming style and documentation conventions to write code that is easily understandable and extensible.
- Explain issues encountered and progress made during a software development project.

In these examples, not all categories of communication skills were explicitly required. For instance, none of the outcomes address teaming or explicitly involve speaking. One of the advantages of distributing communication skills across the curriculum is that it is not required that every class address every skill. For example, teaming might not be desired in lower level programming classes where students must program on their own to master critical skills, and classes taught in large sections will not be able to manage the logistics of students presenting in class. In the former, it is still possible to have students team in a lab setting and in the latter students could still practice speaking in small groups.

Each institution distributes skills across their curriculum in different ways so it would not be practical to produce definitive lists of SLOs for each course involved in this project, but we will produce examples from Miami University and from North Carolina State University as well as instructions on how existing outcomes can be tailored to add communication skills.