University of Dayton
Department of Computer Science
Undergraduate Programs Assessment Plan
DRAFT –September 14, 2011

Department Mission
The Department of Computer Science in the College of Arts and Sciences has as its mission to provide excellent undergraduate and graduate computing degree programs and to support other programs at the University by providing courses that prepare the graduates of those programs to use computing effectively within their discipline. The Department of Computer Science graduates have a thorough knowledge of the fundamentals of computer science and are prepared to continue their education at a graduate school of their choice or to enter the job market able to make immediate contributions in the application of computing and information technology to the benefit of their employer. Graduates are prepared to contribute to the discipline and further their knowledge of computing through active participation in professional societies. The education our graduates receive in computer science, as well as in the humanities and social sciences, fosters the responsible use of computer technology in society. In support of these goals, faculty scholarship serves both to keep faculty current in their knowledge and abilities and to allow student participation in contemporary research projects.

Academic Programs
The department offers three academic programs. Undergraduate students may choose between the Computer Science (CPS) program and the Computer Information Systems (CIS) program. The latter takes a more applied approach to the discipline, allowing the student to choose an approved concentration or minor in another area of study, resulting in an interdisciplinary experience that will prepare the graduate to apply computing knowledge to that area. The CPS program provides more depth in mathematics and science.

At the graduate level, we offer a Master of Computer Science (MCS) degree. However, this document focuses on the assessment of our undergraduate programs.

Foundation Courses
At the heart of both the CPS and CIS curricula are five courses that form the foundation of both programs. These courses are listed below in Table 1. For each of these courses, a list of student learning outcomes will be available. Each semester, faculty teaching these courses will identify student outputs that directly address at least three of the outcomes, and will record student performance on these outputs by determining the percentage of students who achieved each of four performance levels defined below in Table 2. These student outputs may consist of, but need not be limited to, performance on exams, quizzes, and homework assignments that directly address the outcomes. Wherever possible,

multiple measures should be used. Since four of these five courses are required of Computer Engineering (CPE) majors in the School of Engineering (SOE), the five performance levels were chosen to be consistent with those used for SOE's ABET accreditation process, which also requires periodic assessment to ensure continuous improvement of academic programs.

Course Number	Course Title	Bulletin Description
CPS 150	Algorithms and Programming I	Algorithms, programs, and computers. Algorithm development, basic programming and programming structure. Debugging and program verification. Data representation. Computer solutions to numeric and non-numeric problems using a compiler language.
CPS 151	Algorithms and Programming II	Continuation of CPS 150. Emphasis on program design, development and style, string processing, data structures, program modularity, and abstract data type, using a compiler language.
CPS 341	Discrete Structures	Logic and proofs, sets and counting, Boolean algebra, graph theory, directed graphs, mathematical machines, formal languages and grammars.
CPS 350	Data Structures and Algorithms	Advanced concepts of linear data structures, stacks, queues, and abstract data types. Basic and advanced concepts of trees, graphs, hash tables, heaps, algorithm design and analysis techniques.
CPS 346	Operating Systems I	Semaphores, conditions, monitors, and kernels. Concurrent programming, interrupts, memory, and process management. Design and implementation of multithreaded and distributed system components using concurrent languages.

Table 1: Foundation Courses in the CPS and CIS Programs

Performance Level	Description
developing	Student performance is below expectation.
acceptable	Student performance exhibits fundamental understanding of the key concepts.
competent	Student performance indicates a clear and full understanding of the key concepts and can both adequately articulate and implement those concepts.
exemplary	Student performance is outstanding, going beyond expectation.

Table 2: Student Performance Levels

Identified Learning Outcomes

Learning outcomes have already been identified for four of the five foundation courses. It is expected that the learning outcomes for the remaining course (CPS 341) will be identified during the fall 2011 semester, as the semester progresses and as data collection occurs.

Below is a list of the learning outcomes identified for the foundation courses listed. For each, a mapping is provided in parentheses indicating which of the seven University Outcomes[1] are contributed to by the learning outcome. The foundation courses touch on four of the University outcomes – Scholarship, Practical Wisdom, Community, and Vocation. The remaining three outcomes are covered by required study outside of the computer science discipline, including but not limited to courses in the current General Education requirements and the pending Common Academic Program.

Course: CPS 150 – Algorithms and Programming I

In completing the requirements of this course, the student will:

1. Demonstrate the application of basic programming concepts, such as sequential execution of instructions, use of variables and expressions. (Scholarship, Vocation)
2. Apply standard control structures to provide alternate paths of execution. (Scholarship, Practical Wisdom, Vocation)
3. Solve problems using program decomposition through the use of functions. (Scholarship, Vocation)
4. Define classes and objects to represent elements of a problem solution. (Scholarship, Vocation)
5. Develop solutions to problems using object-oriented techniques, and to test and debug those solutions in an integrated development environment. (Scholarship, Practical Wisdom, Vocation)

[1] As defined in the University Assessment Plan (http://assessment.udayton.edu/assessment_plan.htm)

6. Use arrays to store homogenous data, and to design applications for the processing of that data. (Scholarship, Vocation)
7. Document not only what a computer program accomplishes, but how it is accomplished. (Community, Vocation)
8. Express the steps necessary to search and sort data stored in arrays, including the ability to analyze the amount of work necessary to carry out these tasks. (Scholarship, Vocation)

Course: CPS 151 – Algorithms and Programming II

In completing the requirements of this course, the student will:

1. Demonstrate object-oriented concepts, such as data abstraction, information hiding and separation of interface from implementation. (Scholarship, Vocation)
2. Use pointers in C++ to provide indirect access to data objects. (Practical Wisdom, Vocation)
3. Define and implement recursive definitions, structures and algorithms, and be able to apply this to the development of recursive functions in C++. (Scholarship, Practical Wisdom, Vocation)
4. Implement abstract data types that provide well-defined functionality through a clearly documented interface. (Scholarship, Practical Wisdom, Community, Vocation)
5. Apply generic abstract data types generic through the use of templates in C++. (Scholarship, Vocation)
6. Design, implement and test solutions to problems using common data structures, specifically lists, stacks, queues. (Scholarship, Practical Wisdom, Vocation)
7. Analyze the work necessary to perform an algorithm as a function of the amount of data being processed, and to clearly express that analysis in writing. (Scholarship, Community, Vocation)

Course: CPS 350 – Data Structures and Algorithms

In completing the requirements of this course, the student will:

1. Define and construct advanced data structures used in computer applications. These structures include but are not strictly to the following: trees, graphs, and heaps. (Scholarship, Practical Wisdom, Vocation)
2. Design complex data structures as abstract data types, maintaining a clear distinction between implementation and interface, providing proper documentation of both. (Scholarship, Practical Wisdom, Community, Vocation)
3. Define inheritance and information hiding in the design of abstract data types. (Scholarship)
4. Implement and use both linear and advanced data structures in computer applications written in C++. (Scholarship, Practical Wisdom)
5. Design solutions to problems by first determine the data structures that are beneficial in formulating a solution. (Scholarship)
6. Analyze the complexity of algorithms, communicating how the data structures chosen for an application impact that complexity. Collect empirical data and assess the relationship between algorithm complexity and performance. (Scholarship, Practical Wisdom, Community, Vocation)

Course: CPS 346 – Operating Systems I

In completing the requirements of this course, the student will:

1. Use operating system internals such as process creation and management, process scheduling, and memory management (Scholarship, Practical Wisdom, Vocation).
2. Experiment with operating system internals such as process creation and management, process scheduling, and memory management in a modern accessible operating system such as UNIX (Scholarship, Practical Wisdom, Vocation).
3. Demonstrate proficiency in UNIX and C as an operating systems programming language/environment (Scholarship, Practical Wisdom, Vocation).
4. Use the most important system software tools such as compilers (gcc), linkers, loaders, debuggers (gdb), and compilation managers (make) (Scholarship, Vocation).
5. Know UNIX internals, including use of the UNIX kernel system call interface within C programs (Scholarship, Practical Wisdom, Vocation)
6. Design and implement a UNIX shell (i.e., command interpreter) in C (Scholarship, Practical Wisdom, Vocation).
7. Demonstrate various aspects of an operating system such as job and process scheduling and main memory management (Scholarship, Practical Wisdom, Vocation).
8. Develop UNIX Makefiles (Scholarship, Practical Wisdom, Vocation).
9. Demonstrate concurrent programming and synchronization techniques(Scholarship, Practical Wisdom, Vocation).
10. Develop concurrent programs and multi-threaded applications as well as synchronize the control therein (Scholarship, Practical Wisdom, Vocation).
11. Define and discuss concepts of main memory management such as paging and segmentation (Scholarship, Practical Wisdom, Community, Vocation).
12. Define and discuss the fundamentals of distributed systems and computer security (Scholarship, Practical Wisdom, Vocation).

Assessment Instrument and Benchmark

Each semester, faculty members teaching foundation courses will identify a subset of at least three of the learning outcomes identified for the course to be featured in the assessment for the semester. A table will be generated in the format shown below (Table 3) to summarize the performance of the class on the identified learning outcomes. The "Comments" section should be used for additional observations that may not be clear from the table itself, and for observations the faculty members wishes to note regarding how the course went, suggestions for improvements, etc.

Of course, courses differ in rigor and difficulty; so expected performance will vary by course. However, we believe on average 70% of students should perform at the competent or exemplary level, and thus see this as the benchmark to meet or exceed.

Course	CPS 350	Data Structures and Algorithms				
Semester	Example 2011					
Instructor	Dr. Doctor					
Enrollment	18					
			Performance Breakdown (%)			
Learning Outcome	**University Outcomes**	**Evidenced in...**	**developing**	**acceptable**	**competent**	**exemplary**
2. Design complex data structures as abstract data types, maintaining a clear distinction between implementation and interface, providing proper documentation of both.	Scholarship, Practical Wisdom, Community, Vocation	Assignments 1, 2; Exam 2	22%	11%	22%	45%
4. Implement and use both linear and advanced data structures in computer applications written in C++. (Scholarship, Practical Wisdom)	Scholarship, Practical Wisdom	All Assignments	33%	22%	6%	39%
6. Analyze the complexity of algorithms, communicating how the data structures chosen for an application impact that complexity. Collect empirical data and assess the relationship between algorithm complexity and performance. (Scholarship, Practical Wisdom, Community, Vocation)	Scholarship, Community, Vocation	Assignments 4, 5, 6; Exam 1, Final Exam	39%	17%	11%	33%
Comments:	Bifurcated distribution, while not uncommon in this course, is more extreme than usual.					

Table 3: Example Assessment Instrument

Procedures

In order to implement this plan, the following steps will be taken each year:

1. Each semester, faculty teaching foundation courses will submit to the Department Chair their assessment instrument.
2. Each summer, the chair will prepare a brief narrative summary that combined with the assessment instruments will comprise the department's annual assessment report. This summary will include identification of strengths and weaknesses evidenced by the data collected, and a summary of the suggestions for improvements made by the faculty.
3. The chair's narrative summary will also report placement rates of graduates in both positions in industry and graduate schools. This information will be obtained from exit interviews and/or surveys.
4. Each fall, the chair will discuss the annual assessment report at a department meeting.