

Computer Science Course Descriptions

including

Course Outcomes

Assessment Plans

Course Improvements

COURSE DESCRIPTION

Department and Course Com S 227 Coordinators James Lathrop, Steve
Number Fall 2008 Kautz, Pavan Aduri

Course Title Introduction to object-oriented Programming Total Credits 4

Current Catalog Description (3-2) Cr. 4. F.S. An introduction to object-oriented design and programming techniques. Symbolic and numerical computation. Recursion and iteration. Modularity, procedural and data abstraction, specifications and subtyping. Object-oriented techniques. Imperative programming. Emphasis on principles of programming and object-oriented design through extensive practice in design, writing, running, debugging, and reasoning about programs. This course is designed for majors. Credit may not be applied toward graduation for both 207 and 227.

Textbook James Cohoon and Jack Davidson, *Java 5.0 Program Design*

Coordinators James Lathrop, Senior Lecturer,
Steve Kautz, Lecturer
Pavan Aduri, Associate Professor

Course Outcomes

At the end of Com S 227 the students should be able to:

- Write, debug, and document well-structured Java applications of up to 500 lines
- Implement Java classes from specifications
- Effectively create and use objects from predefined class libraries
- Understand the behavior of primitive data types, object references, and arrays
- Use decision and iteration control structures to implement algorithms
- Write simple recursive algorithms
- Use interfaces, inheritance, and polymorphism as programming techniques
- Use exceptions

Relationship between Course outcomes and Program outcomes: B, C

Prerequisite (s) by Topic

High school algebra and geometry

Major Topics covered in the Course

- Managing files; using an integrated development environment
- Objects, classes, methods, fields, and constructors
- Primitive types and references
- Arithmetic expressions, String operations
- Overview of the software engineering process
- Unit testing with the JUnit framework
- Conditional statements and Boolean expressions
- Iteration
- Basic console and text file I/O
- One-and two-dimensional arrays
- Using a symbolic debugger
- Designing with interfaces
- Inheritance, polymorphism, abstract classes
- Exceptions and exception handling

- Recursion
- Searching and sorting

Laboratory exercises

A. Programming projects

Normally five individual-effort projects are assigned throughout the semester, with increasing levels of difficulty.

B. Laboratory activities

1. Files; the Eclipse IDE
2. Java API documentation; the Javadoc tool; creating a simple class
3. String manipulation; evaluating expressions
4. Interacting classes; Boolean expressions
5. Command-line tools and shell commands
6. Code style and formatting
7. The Eclipse debugger
8. Exception handling; File I/O
9. JUnit
10. Inheritance and polymorphism; creating Exception types
11. Recursion

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Write, debug, and document well-structured Java applications of up to 500 lines (B, C)	Evaluation of programming projects	Every semester	Students tend to postpone reviewing and learning new material until just prior to project deadlines.	Implement a series of structured hands-on laboratory exercises for new and/or difficult topics to ensure students absorb them well before needed for projects.
Implement Java classes from specifications (C)	Exams; evaluation of programming projects	Every semester	Students fail to grasp the role of specifications in implementing components of a system.	Make sure that one or more of the programming projects involves integration of student-implemented classes with existing libraries.
Use interfaces, inheritance, and polymorphism as programming techniques (B)	Performance of continuing students at start of Com S 228; exams; evaluation of programming projects	Every semester	Students fail to comprehend the use of interfaces in a design.	Include laboratory exercises using interfaces and inheritance. Ensure that one or more of the programming projects requires students to program with interfaces.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department's undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department's Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Algorithms	_____	_____	Computer Organization and Architecture	_____	_____
Data Structures	_____	_____	Concepts of Programming Languages	<u>2</u>	_____
Software Design	<u>2</u>	_____			

COURSE DESCRIPTION

Department and Course Com S 228 Course Coordinator James Lathrop
Number Fall 2008

Course Title Introduction to Data Structures Total Credits 3

Current Catalog Description (3-1) Cr. 3. F.S. Prereq: C- or better in 227, credit or enrollment in Math 165. An object-oriented approach to data structures and algorithms. Object-oriented analysis, design, and programming, with emphasis on data abstraction, inheritance and subtype polymorphism. Abstract data type specification and correctness. Collections and associated algorithms, such as stacks, queues, lists, trees. Searching and sorting algorithms. Graphs. Data on secondary storage. Analysis of algorithms. Emphasis on object-oriented design, writing and documenting medium-sized programs. This course is designed for majors.

Textbook William J Collins, *Data Structures and the Java Collections Framework, Second Edition*, McGraw Hill 2005. ISBN: 0-07-282379-8

Coordinators James Lathrop, Senior Lecturer

Course Outcomes At the end of ComS 228, the students should:

- be able to write well-structured object-oriented programs of up to 1000 lines of code;
- understand type safety and use of generic types to enforce type safety;
- be able to write programs and class libraries given a specification;
- understand Big-O notation and apply it to simple methods, including methods that utilize complex loops and recursion;
- analyze run-time execution of previous learned sorting methods, including selection and merge sort;
- implement and analyze insertion sort and Quicksort sorting algorithms;
- understand abstract data types and how they are implemented in an object-oriented language;
- understand and implement the List Abstract Data Type (ADT) using both array based and linked-list based data structures, including singly, doubly, and circular linked-lists;
- understand and implement the Stack ADT using both array based and linked-list based data structures;
- understand and implement the Queue ADT using both array based circular queue and linked-list based implementations;
- understand and implement priority based queues;
- understand and implement general tree data structures, including binary tree, both array based and reference based implementations;
- understand and implement binary search trees;
- understand and implement heaps using an array based tree data structure;
- understand and analyze heapsort;
- understand and implement graph data structures;
- understand and implement various algorithms on graph data structures, including finding the minimum spanning tree and shortest path;
- understand and implement hash table data structures and understand the map abstract data type;
- evaluate the runtime of operations such as add and delete on data structures covered in the class.

Relationship between Course Outcomes and Program Outcomes: A, B, C

Prerequisite by Topic

Programming in an object-oriented programming language that include instruction with topics equivalent to those covered in ComS 227.

Credit or enrollment in first semester calculus.

Major Topics Covered in the Course

Topic	Reading from text
CS 227 review	Ch 1, 2
Runtime Analysis	Ch 3
Sorting Algorithms	Ch 5, 11
Collection Interface and Iterators	Ch 4
Lists and the ArrayList class	Ch 6
Lists and linked lists	Ch 7
Stacks and Queues	Ch 8
Trees	Ch 9
Binary Search Trees	Ch 10
Tree Set Interface	Ch 12
Hash Tables	Ch 14
Graphs and Graph Algorithms	Ch 15

Programming Assignments

There are typically 5 programming assignments spaced approximately 3 weeks apart.

Assignment 1: This assignment is comprised of material mostly from ComS 227, but also includes some generic typing. It is designed to be a slightly bigger project than projects completed in ComS 227 with emphasis placed on integrating several different concepts from ComS 227 into a single program. Program documentation, design, and programming style are graded. Unit testing is also reviewed and required in this assignment.

Assignment 2: This assignment gives students practice in designing and implementing various sorting methods, including Quicksort and then measuring the number of CPU cycles required to sort random data sets of various sizes. The students are required to graph their results to understand the relationship between data set size and execution time of the various algorithms. Students are encouraged to write their own test data sets and share with the class.

Assignment 3: In this assignment students must implement a substantial Java interface, typically the List interface. Students are required to code according to the specification and their implementation must match that of one of Sun's implementations. (This allows students to continually check their work against the correct answer.) In order to keep students from downloading source code and not learning how to manipulate linked list or array lists, restrictions are placed on the implementation. For example, instead of implementing a list using a standard doubly linked-list, the implementation must create list nodes that skip 2 forward and 3 back.

Assignment 4: This assignment typically requires students to write a complete program that requires manipulating a tree structure. Past programs include writing a compression and decompression methods for the Lempel-Ziv compression algorithm, implementing a simple interpreter for a made-up programming language, and implementing a trinary search tree. It is encouraged that instructors try to create an assignment that encompasses and solves some aspect of a real-world problem.

Assignment 5: This assignment consists of two parts, one mandatory and one optional. The mandatory part tests students ability to implement a graph structure and compute some properties of the graph such as shortest path or connectedness. In addition, the students are given an interface that allows them to control a virtual robot that roams the galaxy by traversing wormholes in a game environment. Systems and wormholes form a weighted directed graph and thus graph algorithms may be useful in defeating the opponent. Students are required to write a basic robot that wins against inept computer controlled robot. In addition, students may optionally enter a tournament where their robots are played against each other on the last day of class.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Write, debug, and document well-structured Java applications of up to 1000 lines	Evaluation of programming projects	Every semester	Students are surprised by the difference in size in the programming projects required in Com S 228 and ComS 227. This leads to students starting assignments late and not finishing them.	Assignments are given early to students with explanations on how long the assignment can take. Electronic bulletin boards are used for student discussion of assignment.
Implement Java classes from specifications	Exams; evaluation of programming projects	Every semester	Students fail to grasp the role of specifications in implementing components of a system.	Assignment 3 requires students to completely implement the List interface in the Java collection package using an uncommon backing data structure. Other assignments include significant specification that require students to read and understand them in order to complete the project with a good score.
Lack of retention of prerequisite material.	Exams, programming projects, and student class interaction.	Every semester	Students need to understand concepts from ComS 227 in order to do well in ComS 228.	Two days of lecture devoted to reviewing material from Com S 227 with emphasis placed on topics the students themselves believe they do not understand. The first assignment includes only a few new ideas together with requirements that force students to use much of the topics covered in ComS 227.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Algorithms	<u>0.5</u>	_____	Computer Organization and Architecture	_____	_____
Data Structures	<u>2</u>	_____	Concepts of Programming Languages	_____	_____
Software Design	<u>0.5</u>	_____			

COURSE DESCRIPTION

Department and Course Com S 229 Course Coordinator James Lathrop
Number Fall 2008

Course Title Advanced Programming Techniques Total Credits 3

Current Catalog Description (3-0) Cr. 3. F.S. Prereq: 228, credit or enrollment in Math 166. Object-oriented programming experience using a language suitable for exploring advanced topics in programming. Topics include memory management, parameter passing, inheritance, compiling, debugging, and maintaining programs. Significant programming projects.

Textbook (optional) Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley 2000. ISBN 0201700735. Nicolai M. Josuttis, *The C++ Standard Library (A Tutorial and Reference)*, Addison-Wesley 1999, ISBN: 0201379260. Kernighan and Ritchie, *The C Programming Language* second edition, Prentice Hall, 1998. ISBN: 0-13-110362-9

Coordinators James Lathrop, Senior Lecturer

Course Outcomes At the end of ComS 229, the students should:

- be able to write well-structured complex programs from specifications that may not be precise in OO and non OO programming languages,
- communicate questions to clarify program requirements,
- understand differences between managed and unmanaged programming languages such as Java and C++ or C.
- understand simple build systems and create software projects that utilize a simple build system to produce several target executable programs,
- understand how to read and program using the C programming language.
- understand and utilize memory management techniques for C programming, including allocation and tracking of dynamically allocated memory,
- program and link together elements of Java programs and C programs using the Java native Interface,
- compare and contrast elements of C programming language (a non OO language) and the Java programming language,
- understand how to read and program in the C++ programming language,
- understand and use C++ templates and the standard template library,
- understand memory management techniques used in C++ programming,
- understand memory management techniques used in managed languages such as Java such as mark and sweep garbage collection algorithms,
- understand and program basic elements of a GUI interface.

Relationship between Course Outcomes and Program Outcomes: A, B, C

Prerequisite by Topic

Programming in an object-oriented programming language that include instruction with topics equivalent to those covered in ComS 227.

Understanding of data structures with topics equivalent to those covered in ComS 228.

Credit or enrollment in second semester calculus.

Major Topics Covered in the Course

Introduction
 The C Programming Language
 Compiling C programs
 Difference between C and Java
 The make program
 Java GUI concepts using Swing
 Java JNI and the C programming language
 The C++ programming language
 Compiling C++ programs
 Differences in C++, Java and C
 C++ templates and the STL
 Memory management
 Multiprogramming and threads in Java and C++

Programming Assignments

There are typically 2 large programming assignments during the semester.

Assignment 1: Assignment 1 includes aspects that requires the student to program a significant project in the C programming language with interfaces with the Java programming language. Topics required to complete project include C programming, C memory management, Java JNI interface, the make system, and correctly compiling C programs (using the make system). Extensive use of data structures and concepts from ComS 227 and ComS 228 are required in order to satisfactorily complete the assignment.

Assignment 2: Assignment 2 includes aspects that requires the student to program a significant project in C++. The student may be required to interface with C and Java to complete the assignment. Use of templates and data structures using templates may be required.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
ALL	Evaluation of class at end of the semester	Every semester	Students would benefit greatly from a recitation (even optional) for this class so that there is time set aside other than lecture where students can get help.	Addition of recitation section to ComS 229. (not yet implemented)
Design and program well-structured complex programs	Evaluation of programming projects	Every semester	Students have difficulties designing and assembling a large complex project given a list of complex interrelated requirements.	Complex assignment requirements are presented as in organized units for the students with separate implementations that can then be linked together in the final project.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department's undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department's Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Algorithms	<u>1.0</u>	<u> </u>	Computer Organization and Architecture	<u> </u>	<u> </u>
Data Structures	<u>1.0</u>	<u> </u>	Concepts of Programming Languages	<u>0.5</u>	<u> </u>
Software Design	<u>0.5</u>	<u> </u>		<u> </u>	<u> </u>

COURSE DESCRIPTION

Department and Course Cpr E 281 Course Coordinator Ahmed Kamal
Number Fall 2008

Course Title Digital Logic Total Credits 4

Current Catalog Description (3-2) Cr. 4. F.S. Prereq: sophomore classification. Number systems and representation. Boolean algebra and logic minimization. Combinational and sequential logic design. Arithmetic circuits and finite state machines. Use of programmable logic devices. Introduction to computer-aided schematic capture systems, simulation tools, and hardware description languages. Design of a simple digital system.

Textbook S. Brown and Z Vranesic, *Fundamental of Digital Logic with Verilog Design*, McGraw-Hill, Second Edition, 2008.

Coordinator Ahmed Kamal, Professor of Electrical and Computer Engineering

Course Outcomes

- To introduce number systems and codes, and digital representation of data, and to teach the general concepts in digital logic design, including logic elements, and their use in combinational and sequential logic circuit design.
- Students will also be introduced to computer-aided schematic capture systems, simulation tools and hardware description languages, and will use programmable logic devices.

Relationship between Course outcomes and Program outcomes: A, C

Prerequisites by Topic

Major Topics covered in the Course

- 1. Introduction** (Chapter 1)
 - Digital Hardware
 - The design process
 - The design of digital hardware
- 2. Introduction to Logic Circuits** (Chapter 2)
 - Variables and functions
 - Inversion
 - Truth tables
 - Logic gates and networks
 - Boolean algebra
 - Synthesis using AND, OR and NOT gates
 - NAND and NOR logic networks
 - Design examples
- 3. Implementation Technology** (Chapter 3)
 - Transistor switches
 - NMOS logic gates
 - CMOS logic gates
 - Negative logic systems
 - Standard chips
 - Programmable devices
- 4. Optimized Implementation of Logic Functions** (Chapter 4)
 - Karnaugh map
 - Strategy for minimization
 - Minimization of product-of-sums forms
 - Incompletely specified functions
 - Multiple-output circuits

- Multilevel synthesis
- Analysis of multilevel circuits
- 5. Number Representation and Arithmetic Circuits** (Chapter 5)
 - Positional number representation
 - Additional unsigned numbers
 - Signed numbers
 - Fast adders
 - Multiplication
 - Other number representations
 - ASCII character code
- 6. Combinational-Circuit Building Blocks** (Chapter 6)
 - Multiplexers
 - Decoders
 - Encoders
 - Code converters
 - Arithmetic comparison circuits
- 7. Flip-flops, Registers, Counters and a Simple Processor** (Chapter 7)
 - Basic latch
 - Gates SR latch
 - Gated D latch
 - Master-slave and edge-triggered D flip-flops
 - T flip-flops
 - JK flip-flops
 - Registers
 - Counters
 - Reset synchronization
- 8. Synchronous Sequential Circuits** (Chapter 8)
 - Basic design steps
 - State assignment problem
 - Mealy state model
 - Serial adder example
 - State minimization
 - Design of a counter using the sequential circuit approach
 - FSM as an arbiter circuit
 - Analysis of synchronous sequential circuits
 - Algorithmic state machine
 - Formal model for sequential circuits
- 9. Digital system Design** (Chapter 10)
 - Building block circuits
 - Design examples
 - Clock synchronization

Important Notes

- 1. Requesting academic accommodation:**
Please address any special needs or special accommodations with me at the beginning of the semester or as soon as you become aware of your needs. Those seeking accommodations based on disabilities should obtain a Student Academic Accommodation Request (SAAR) form from the Disability Resources (DR) office (515-294-7220). DR is located in 1076 Student Services Building.
- 2.** All class material, including copies of lecture notes, homeworks and their solutions, labs, quiz solutions, and midterm exam solutions will be posted on WebCT. Please **check WebCT regularly**.
- 3.** Homeworks are due at the **beginning of class**.
- 4.** Quizzes will be at the **end of class**.
- 5.** All quizzes and exams will be held in the **classroom**.
- 6. Pre-Labs** must be submitted to the TA at the **beginning of the lab**.
- 7.** There are no supplementary or make-up exams in this course.
- 8.** There will be no changes in the exams, labs, homeworks and quiz dates. Please refer to the **Schedule** for those dates.

9. No late homework assignments will be accepted.
10. Students may discuss the homeworks, and how to solve them. Students are encouraged to use the Discussion Forum on WebCT (I will be monitoring the forum and will post contributions if needed). However, the actual solution of the homeworks must be done independently. Students found to be copying homework assignments, or project codes will be subject to the ISU cheating and plagiarism regulations.
11. The instructor and TAs are available to help you during the **office hours**. Appointments outside the office hours can also be arranged. Please contact the instruction or the TAs to arrange for appointments.
12. All pagers and cellular telephones **must be turned off** during the class time.

Thank you for your cooperation.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Comprehension of tradeoffs involved in design choices (H)	Student performance on quizzes and exams which are graded by the course instructor and not the TAs, senior outcomes assessment surveys	Every semester	Students had difficulties in formulating problems in computer architecture and thus could not solve them.	- More in class examples are worked out. - Periodic in-class activities are conducted where students work in groups of two or three to formulate and solve problems in lecture.
Comprehension of memory tradeoffs and management policies	Performance on quizzes and homework	Every semester	Students seemed to have different learning strategies for these concepts	A visual hands-on Java tutorial is available for students who are “visual learners.”

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Algorithms	_____	_____	Computer Organization and Architecture	<u> 2 </u>	_____
Data Structures	_____	_____	Concepts of Programming Languages	<u> 1 </u>	_____
Software Design	_____	_____			

COURSE DESCRIPTION

Department and Course Com S 309 Course Instructor Simanta Mitra
Number Fall 2008

Course Title Software Development Practices Total Credits 3

Current Catalog Description (3-1) Cr. 3. F.S. Prereq: Com S 228 with C- or better, Com S 229 or Cpr E 211, Engl 250. A practical introduction to methods for managing software development. Process models, requirements analysis, structured and object-oriented design, coding, testing, maintenance, cost and schedule estimation, metrics. Programming projects. Nonmajor graduate credit.

Textbook Eric J. Braude, *Software Engineering: An Object Oriented Perspective*, John Wiley and Sons Inc., ISBN 0-471-32208-3.

Reference

- Software Engineering, Ian Sommerville, Addison-Wesley, 6th edition, 2001.
- *An Integrated Approach to Software Engineering* by Pankaj Jalote, 2nd Edition.
- *Software Engineering* by Shari Pfleeger.
- *UML Toolkit* by Hans-erik Eriksson and Magnus Penker.

Coordinators Robyn Lutz, Professor
Simanta Mitra, Senior Lecturer

Course Outcomes To introduce the students to the major software engineering topics and position them to lead medium-sized software projects in industry.

1. Students will learn to work as a team and to focus on getting a working project done on time with each student being held accountable for their part of the project.
2. Students will learn about risk management and quick prototyping to de-risk projects.
3. Students will learn about and go through the software development cycle with emphasis on different processes - requirements, design, and implementation phases.
4. Students will learn details about different artifacts produced during software development.
5. Students will learn about different software development process models and how to choose an appropriate one for a project.
6. Students will gain confidence at having conceptualized, designed, and implemented a working, medium sized project with their team.

Relationship between Course Outcomes and Program Outcomes: A, B, C, D, E, F, G, H, I, J, K

Prerequisites by Topic

1. A high-level programming language.
2. Data structures.
3. Proficiency in written English.

Major Topics covered in the course

1. Intimate familiarity with various software artifacts (documents/code) produced during a software development project.
2. Top Issues in Software Engineering
3. Project Management: Process Models, Planning, Scheduling, Cost Estimation, Risk Management, Metrics, Project execution, Project termination.
4. Software Requirements: Elicitation, Specification, Verification, Validation.
5. Software Architecture: Decomposition, Modularity, Specification of Interfaces, Design tradeoffs, Design Evaluation.
6. Configuration Management: Use of source control tools, change management.
7. Testing techniques, Inspections/Reviews.

Laboratory Projects

Semester project introduced first week, due last week: 16 weeks.

One large software development project lasting the entire semester. Artifacts developed include:

- Project proposal
- Screenshots/Interface prototyping
- SRS-1 (Customer Requirements)
- SRS-2 (Developer Requirements)
- DD-1 (Design Decomposition into modules + processes/threads)
- DD-2 (Module interfaces)
- DD-3 (Design rationale, Design evaluation)
- Test Plan
- Lessons Learnt from project.
- Working Project Code and Test Codes.
- Project Poster

Team participation is carefully monitored throughout the semester in a variety of ways:

- Minutes of each meeting are collected using a template.
- Artifact Work distribution form is collected using a template.
- Source Control tools are used to gather information on what each team member worked on, on what dates etc.
- During final demo and also in-class status and prototype demonstrations, each team member is grilled on his/her specific contribution.

Teams have to maintain a portfolio of these artifacts (both hardcopy and softcopy under revision control).

There are several project status update and prototype demonstration sessions throughout the semester. There is a final project demonstration to the TAs and also a Best Project awards session where the best teams show off their project to the class.

Each artifact goes through two rounds of submission. In the first round, feedback is provided. In the second round, a final grade is provided.

Assessment Plan for the Course

This course is unique (given that it is a project course) in that it has an impact on each of the Program Learning outcomes – from A to I!

These outcomes are measured/improved several times during the semester via:

- in-class quizzes
- in-class group activities
- in-class prototype and design presentations

- in-class question/answer and discussion sessions
- Interviews with individual teams about design
- Project demonstrations
- Having students explain their contribution and lessons learnt
- The project artifacts including minutes of meetings and work-distribution forms.

Several improvements were identified and implemented – and each of these applies to multiple program outcomes.

Outcome	How Measured	When Measured	Improvements identified	Improvements Implemented
Team Work (course outcome #1)	1) Work distribution forms that students have to submit along with every artifact. 2) Code logs from subversion source control tool.	At each submission and also at the end of the semester.	Team member problems: <ul style="list-style-type: none"> - some drop - some don't work at all - some don't do coding - some don't do docs 	A huge emphasis is also placed on how important team work is towards success of projects. <ul style="list-style-type: none"> • Team work etiquette is explained. • Minutes of meetings are required! • For each team submission (there are 10 of them), work distribution forms have to be submitted. • Teams are encouraged to look out for warning signs of bad team work. • Source control tool (Subversion) is used from early in the semester and monitored by the TAs and the instructor for signs of some students not working. • Accountability is drummed in. Students have to mark which parts of each document was created by them. Similarly, source control tools indicate which parts of code are developed by a specific student. • In the case – that some students drop – modular design – helps to minimize the impact. Poor workers are given peripheral pieces – so that if they do not complete their parts – the entire project is not jeopardized.
Risk Management (course outcome #2)	Project artifact submission	Several times during the semester (at maximum two weeks intervals)	Many times, teams do not get started early enough – and then the quality of the software artifacts suffers. In addition, they are faced with a lot of word during	The entire documentation work is broken up into several increments. For example, the SRS is broken into SRS-1 and SRS-2. Similarly, the SDD is broken into SDD-1, SDD-2, and SDD-3. With about two weeks for each document, students have to keep working on them throughout the

Outcome	How Measured	When Measured	Improvements identified	Improvements Implemented
			the dead week – and this affects their grades in other courses as well.	<p>semester and this helps to create better documents.</p> <p>In terms of coding, a huge emphasis is put on early prototyping – right from day one! Students are encouraged to create small working fragments of code and play with networking and threading and databases and GUI etc. This way – the major technical risks of projects are encountered early and then can be addressed early on.</p> <p>The demos are scheduled the week BEFORE the dead week. This forces students to start work real early. Also – it then gives them breathing time during the dead week to focus on other courses and also the final exam.</p>
Experiencing software development lifecycle processes (course outcome #3)	<p>Project artifact submission</p> <p>Final project report with lessons learnt.</p>	Several times during the semester (at maximum two weeks intervals)	Doing the project + all the documentation – is a LOT of work and sometimes teams seem jaded and overworked – just by the thought of it.	<p>Maintaining enthusiasm and motivation is an important aspect of projects like this. Several things are done towards this:</p> <ul style="list-style-type: none"> • The semester is started with a display of posters from previous semesters. These full-size posters have information about the features, the design, lessons learnt, and students pictures. There is a lot of excitement right from the very start. At the end of the semester – students create their own posters and this is added to the set of existing posters. • Healthy competition between teams is encouraged. A best team project award and a best documentation award is given at the end of the semester. These are certificates and signed by the department chair and the course instructor. • There is a best projects presentation at the end of the semester - where some faculty members are also invited to attend. Students get to vote for the best project in their eyes – although the final

Outcome	How Measured	When Measured	Improvements identified	Improvements Implemented
				<p>determination is made by the instructor.</p> <p>Another important improvement is to lessen the amount of documentation – while still teaching the important concepts in creating the SRS and SDD documents. This is achieved by limiting the work to be done by each student (for example – each student needs to elaborate and write detailed requirements for only two features).</p>
Learn details of software artifacts (course outcome #4)	Design documents	Middle of semester, and End of semester.	Poor quality Architectural documents were being submitted.	At this level, students had never designed anything. They had no idea of architecture. It was thus impossible for them to submit good architectural docs before coding. Instead, I started to have them complete the final architectural documentation AFTER all code is done – allowing them to reverse-engineer and refactor their code. This experience will allow them to create better architectural documents in the future also.
Learn about process models (course outcome #5)	questions in class + final exams	Several times during semester, and End of semester	Learning about processes and process models is sometimes tedious – for specially students without internship experiences.	A lot of time is spent on explaining why it is important to learn about these. For example, following best practices and using tools such as defect tracking and issue tracking – is important for auditing purposes and to establish that due diligence was followed – and that can help in legal issues. Examples from industry is used to motivate students.
Good working projects at end of semester (course outcome #6)	in-class presentations during semester project demo at end of semester	Several times during semester, and End of semester	Team projects were either too ambitious or too simple. Either case is detrimental to the student’s learning experience in the class.	Teams have to submit three proposals along with their evaluations about the complexity of each project. The instructor does a separate evaluation and either adds features or trims features to make the projects comparable in terms of difficulty and size. Project complexity evaluation is an ongoing process – and the instructor and TAs keep tabs on the project throughout the semester – from requirements, prototyping, design, and final demo – with feedback being given at each phase.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department's undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department's Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	_____
Software Design	<u> 3 </u>	_____			

COURSE DESCRIPTION

Department and Course Com S 311 Course Instructor Giora Slutzki
Number Fall 2008

Course Title Design and Analysis of Algorithms Total Credits 3

Current Catalog Description (3-1) Cr. 3. F.S. Prereq: 228 with C- or better, 229 or Cpr E 211, Math 166, Engl 250, and either Com S 330 or Cpr E 310. Basic techniques for design and analysis of efficient algorithms. Asymptotic ,worst-case, analyses. Graphs and graph algorithms. Design techniques such as greedy (scheduling, Dijkstra's algorithm, minimum spanning trees, Huffman codes), divide-and-conquer (mergesort, integer multiplication), and dynamic programming (segmented least-squares, knapsack, sequence alignment, shortest paths). Network flows, Ford-Fulkerson algorithm. NP-completeness. Programming project. Credit may not be applied toward graduation for both 311 and 381. Nonmajor graduate credit.

Textbooks (1) *Algorithm Design* by J. Kleinberg and E. Tardos; Addison Wesley, 2006, ISBN 0-321-29535-8
(2) *Introduction to Algorithms* by T.H. Cormen, C.E. Leiserson, and R. Rivest; McGraw-Hill, 2nd edition, 2001. ISBN 0-07-013151-1

Coordinators Giora Slutzki, Professor
David Fernandez-Baca, Professor

Course Outcomes Algorithms are the basic recipes for solving computational problems. Upon completion of this course, students should:

- understand the fundamental principles underlying algorithm analysis and design and be able to apply them in specific instances
- understand asymptotic worst-case analysis
- understand essential algorithm design techniques such as divide-and conquer, dynamic programming and the greedy methods and many of its applications
- understand network flow algorithms and applications
- understand the notion of NP-completeness.

Relationship between Course Outcomes and Program Outcomes: A, J

Prerequisite by Topic

1. Data Structures
2. Discrete Computational Structures

Major Topics covered in the Course

1. Asymptotic analysis of algorithms
2. Recurrences
3. Sorting
4. Data Structures
5. Advanced Algorithm Design Techniques
6. Graph Algorithms
7. Network Flow Algorithms
9. NP-Completeness

Laboratory Projects

1. Implementation of simple kind of suffix trees and operations on them

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Understanding of some basic algorithmic principles (H)	Performance on homework and exams, senior outcomes assessment surveys	Every semester	Students had difficulties in understanding greedy algorithms vs., dynamic programming and strongly connected components.	More examples were given in lecture to illustrate the execution of algorithms.
Designing computer programs for abstract algorithms (H)	Extra credit programming project, senior outcomes assessment surveys	Every semester	More practical problems for students to solve using algorithms learned in lecture.	Some improvements implemented but more room for improvement.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department's undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department's Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

As indicated in the tables in Section 5-B, the credits for this course fall in the advanced category, because of the rigor and depth of the topics covered. The text book by Cormen et al. is typically used as a graduate-level text book in many universities.

	CORE	ADVANCED		CORE	ADVANCED
Algorithms	_____	3	Computer Organization and Architecture	_____	_____
Data structures	_____	_____	Concepts of Programming Languages	_____	_____
Software Design	_____	_____			

COURSE DESCRIPTION

Department and Course Com S 321 Course Coordinator Gurpur Prabhu
Number Fall 2008

Course Title Introduction to Computer Architecture and Total Credits 3
Machine-Level Programming

Current Catalog Description (3-1) Cr. 3. F.S. Prereq: 229, Cpr E 210 and Engl 250. Introduction to computer architecture and organization. Emphasis on evaluation of performance, instruction set architecture, datapath and control, memory-hierarchy design, and pipelining. Assembly language on a simulator. Nonmajor graduate credit.

Textbook D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufmann Publishers Inc., Second edition.

Reference Interactive Computer Architecture Tutorial by Gurpur Prabhu
<http://www.cs.iastate.edu/~prabhu/Tutorial/title.html>

Coordinator Gurpur Prabhu, Associate Professor

Course Outcomes

- Students will learn about computer performance, computer design, and tradeoffs between cost and performance as well as between hardware and software
- Students will formulate and solve problems, understand the performance requirements of systems
- Students will learn to communicate effectively and learn to think creatively and critically, both independently and with others

Relationship between Course outcomes and Program outcomes: B, C, J

Prerequisites by Topic

1. A High-level Programming Language
2. Introduction to Digital Design

Major Topics covered in the Course

1. Introduction: Computer abstractions and technology
2. Measuring and Evaluating Performance: Metrics of performance; Amdahl's Law; Cost/benefit tradeoffs; Instruction count, CPI, Clock cycle time; Comparing and summarizing performance
3. Instruction Set Architecture and Assembly Language of MIPS; Signed and unsigned numbers; Representing instructions - the MIPS instruction set; Instructions for decision making; Supporting procedures - stack pointer; Arithmetic and Logical operations
4. Processor Datapath and Control; Building a datapath for R-type, I-type, and Jump instructions; A simple implementation scheme; How control is supposed to work; A multicycle implementation
5. Memory Hierarchy Design: A framework for memory hierarchies; The basics of caches; Placement, replacement, and memory interaction policies; Measuring and improving cache performance; FIFO, LRU, write back and write through policies
6. Enhancing Performance with Pipelining; Overview of pipelining; A pipelined datapath; Data hazards and forwarding; Data hazards and stalls; Branch hazards
7. Exceptions and Exception Handling: Interrupts and exceptions; Hardware/software interface; Interrupt handlers

Laboratory Projects

5 MIPS Assembly Language Programming Exercises, Lab Exam as part of Exam 2

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Comprehension of tradeoffs involved in design choices (H)	Student performance on quizzes and exams which are graded by the course instructor and not the TAs, senior outcomes assessment surveys	Every semester	Students had difficulties in formulating problems in computer architecture and thus could not solve them.	- More in class examples are worked out. - Periodic in-class activities are conducted where students work in groups of two or three to formulate and solve problems in lecture.
Comprehension of memory tradeoffs and management policies	Performance on quizzes and homework	Every semester	Students seemed to have different learning strategies for these concepts	A visual hands-on Java tutorial is available for students who are “visual learners.”

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Algorithms	_____	_____	Computer Organization and Architecture	<u>2</u>	_____
Data Structures	_____	_____	Concepts of Programming Languages	<u>1</u>	_____
Software Design	_____	_____			

COURSE DESCRIPTION

Department and Course Com S 330 Course Instructor Soma Chaudhuri
Number Fall 2008

Course Title Discrete Computational Structures Total Credits 3

Current Catalog Description (3-1) Cr. 3. F.S. Prereq: 229, Math 166 and Engl 250. Concepts in discrete mathematics as applied to computer science. Logic, proof techniques, set theory, relations, graphs, combinatorics, discrete probability and number theory. Nonmajor graduate credit.

Textbook Discrete and Combinatorial Mathematics (Fifth ed.) by *Ralph P. Grimaldi*

Coordinators Giora Slutzki, Professor
Jack Lutz, Professor
Soma Chaudhuri, Associate Professor

Course Outcomes

To introduce students to the theoretical foundations of computer science which prepare them for their study of virtually all subsequent material in the curriculum. Upon completing this course the students should be able to:

- define the fundamental discrete mathematical structures used in computer science
- reason about these structures and prove rigorously that their reasoning is correct
- apply them in problem solving and analysis, and
- know basic problem solving strategies and be adept at using them

Relationship between Course Outcomes and Program Outcomes: A, J

Prerequisite by Topic

1. Two semesters of Programming
2. Two semesters of Calculus

Major Topics covered in the Course

1. Sets, Functions, Asymptotics
2. Number Theory
3. Logic
4. Proof Methods
5. Combinatorics
6. Discrete Probability
7. Recursive Algorithms and Recurrences
8. Relations
9. Graph Theory

Laboratory Projects

None.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Reason about structures and prove their correctness	Performance on homework and exams	Every semester	Students have difficulties in writing proofs	Demonstrate proof techniques through examples and show students how to generalize and produce arguments to cover the cases in the examples.
Be adept at using problem-solving strategies	Feedback from Recitation Section	Every semester	Some students are not able to formulate strategies for new problems.	Introduce students to the technique of reduction – where new problems are reduced to old ones and then apply strategies used to solve the old problems.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

As indicated in the tables in Section 5-B, the credits for this course fall in the mathematics category. This has been the designated category for this course since 1994.

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	_____
Software Design	_____	_____	Mathematics	<u>3</u>	_____

COURSE DESCRIPTION

Department and Course Number Com S 331 Course Coordinators Giora Slutzki,
Jack Lutz,
Soma Chaudhuri

Course Title Theory of Computing Total Credits 3

Current Catalog Description (Cross listed with Ling.) (3-1) Cr. 3. F.S. Prereq: 228, 229, Com S 330 or CprE 310, Math 166, and Engl 250. Models of computation: finite state automata, pushdown automata and Turing machines. Study of grammars and their relation to automata. Limits of digital computation, unsolvability and Church-Turing thesis. Chomsky hierarchy and relations between classes of languages. Nonmajor graduate credit.

Textbook *Automata and Computability* by Dexter Kozen

Reference *Elements of the Theory of Computation*, by Harry Lewis and Christos Papadimitriou (2nd edition)

Introduction to the Theory of Computation, by Michael Sipser

Introduction to Automata Theory, Languages, and Computation, by John Hopcroft, Rajeev Motwani and Jeff Ullman.

Coordinators Giora Slutzki, Professor
 Jack Lutz, Professor
 Soma Chaudhuri, Associate Professor

Course Outcomes

- To introduce the computer science students to the theoretical foundations of computer science.
- To study abstract models of information processing machines and limits of digital computation.
- To provide theoretical preparation for the study of programming languages and compilers.
- To develop the skills of formal and abstract reasoning as needed; for example, when designing, analyzing, and/or verifying complex software/hardware systems.

Relationship between Course Outcomes and Program Outcomes: A, J

Prerequisite by Topic

1. Programming with Data Structures
2. Discrete Computational Structures

Major Topics covered in the Course

1. Introduction. Alphabets and strings, concatenation, languages, operations on strings and languages, regular expressions and regular languages.
2. Finite-state Automata. Deterministic finite-state automata, non deterministic finite-state automata, subset construction, regular expressions, Kleene's theorem.
3. Push-down Automata. Non deterministic push-down automata, acceptance by final state and empty stack, deterministic push-down automata.
4. Turing Machines. Deterministic Turing machines, recursive and recursively enumerable languages, non deterministic Turing machines, always-halting non deterministic Turing machines.
5. Grammars and Their Machines. Grammars: regular grammars context-free grammars, phrase-structure and contest-sensitive grammars; transitions between grammars and machines. Derivations and derivation trees. Simplification of context-free grammars and Chomsky normal

- form. The Chomsky hierarchy.
- 6. Properties of Classes languages. Boolean closure properties, regular closure properties, other closure properties.
- 7. Properties of languages. Pumping properties of languages and non-membership proofs for regular and context-free languages.
- 8. Other topics – to be decided as time permits.

Laboratory projects

None.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
To develop skills of formal and abstract reasoning (A, H)	Student performance on homework and exams	Every semester	Some students have difficulty extracting useful methods from proofs.	For long proofs, an overview is given in one lecture; the overview is repeated before the proof in the next lecture; then the technique is applied to an example with <i>explicit</i> reference to the proof.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

As indicated in the tables in Section 5-B, the credits for this course fall in the advanced category, because of the rigor and depth of the topics covered.

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	_____
Software Design	_____	_____	Theory	_____	3

COURSE DESCRIPTION

Department and Course Com S 342 Course Hridesh Rajan
Number Coordinators Ting Zhang

Course Title Principles of Programming Languages Total Credits 3

Current Catalog Description (3-1) Cr. 3. F.S. Prereq: 321, 330 or Cpr E 310, either 309, 362 or 363, Engl 250. Organization of programming languages emphasizing language design concepts and semantics. Study of language features and major programming paradigms, especially functional programming. Programming projects. Nonmajor graduate credit.

Textbooks:

- Essentials of Programming Languages (3rd Edition) by Daniel P. Friedman and Mitchell Wand (MIT Press 2008)
- Programming Language Pragmatics (3rd Edition) by Michael L. Scott (Morgan Kaufmann, 2009).

Reference

- The Little Schemer (Fourth Edition) by Daniel P. Friedman and Matthias Felleisen (MIT Press, 1996).
- Effective Java (2nd Edition) by Joshua Bloch (Prentice Hall, 2008).
- Learning Python (3rd Edition) by Mark Lutz (O'Reilly, 2008)

Coordinators

- Hridesh Rajan, Assistant Professor
- Ting Zhang, Assistant Professor

Course Outcomes

The main objective is that students will have a deep, working knowledge of the functional paradigm and the key ideas used in modern programming languages. In more detail the essential objectives for this course are that students will be able to:

- Write and modify programs using a mostly-functional style. This means programming that makes effective use of the abstraction mechanisms of functional languages, such as higher-order functions (functions that take functions as arguments and return functions as results) to achieve generality and abstraction.
- Write and modify programs that make effective use of data abstraction.
- Modify interpreters to change or enhance their behavior so as to implement various features of programming languages such as: control flow, variables, recursion, scoping, syntactic sugars, arrays, parameter passing mechanisms, type checking, objects, classes, and inheritance.
- Write programs using such features, and explain, using appropriate terminology, the user-visible behavior of such programs.
- Explain, using appropriate terminology, the data structures and algorithms used in interpreters to implement such features. Compare alternatives in the design and implementation of such features.

Relationship between Course Outcomes Program Outcomes: C, H, I, J, K

Prerequisite(s) by Topic

1. Computer architecture and machine-level programming
2. Theory of computing
3. File organization and processing

Major Topics covered in the Course

1. Formal language theory: regular languages, context-free languages, finite state automata
2. Write and modify programs in functional style, introduction to languages such as Scheme and Haskell and language implementation techniques such as lazy data structures and monads
3. Make effective use of data abstraction
4. Learn about language constructs by changing or enhance interpreters to have features such as: control flow, variables names and bindings, recursion, static and dynamic scoping, syntactic sugars, arrays, parameter passing mechanisms, continuations, co-routines, type checking, objects, and inheritance
5. Write programs using such features and explain their behavior
6. Explain the data structures and algorithms used in interpreters
7. Logical programming: Prolog, resolution principles, unification algorithm, search with backtracking, cut operation, negation as failure
8. Compare alternatives in programming language design and implementations

Laboratory projects

1. Using Scheme, Python, Prolog, Haskell
2. Language Design and Recursion: LL(1) parsing
3. Formulating Abstractions with Higher-order Procedures: Curry and Uncurry functions
4. Data Abstraction and Object-oriented techniques
5. Conventional interfaces and symbolic data: list manipulations
6. Java visitors and iterators, multiple representations, and systems with generic operations

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
An ability to design, implement, and evaluate a computer-based system, process, component or program to meet desired needs.	Student performance on homework, projects, and exams Feedback from companies	Every semester	Students had an imperative method of approaching a problem.	Carefully stressed the importance of other computing paradigms such as logic and functional paradigms
An ability to engage in continuing professional development.	Student performance on homework and exams Feedback from companies	Every semester	Students didn't have a basic framework for learning new computer languages.	Taught a module that lays the framework
An ability to use current techniques, skills, and tools necessary for computing practices.	Student performance in homework and exams Student feedback during graduating	Every semester	Students didn't have much chance to apply their knowledge about PL theory to new mainstream language	Included an exercise to study current languages such as Python, C#, and Java 1.7

	senior assessment		proposals.	
An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems that demonstrate comprehension of the tradeoffs involved in design choices.	Student performance in homework and exams.	Every semester	Formal language theory and recursive (top-down/bottom-up) construction of functions.	Included a module on LL(1) parsing to address this issue and basic preview module on regular languages.
An ability to apply design and development principles in the construction of software systems of varying complexity.	Student performance on programming homework, and projects.	Every semester	Students didn't know about functional and logic programming-based design styles.	Taught the "follow the grammar" principle which helps with writing functional programs.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department's undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department's Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

As indicated in the tables in Section 5-B, the credits for this course fall in the advanced category, because of the rigor and depth of the topics covered.

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	2
Software Design	_____	1			

COURSE DESCRIPTION

Department and Course Com S 352 Course Instructor Lu Ruan
Number Fall 2008 Dimitris Margaritis

Course Title Introduction to Operating Systems Total Credits 3

Current Catalog Description (3-1) Cr. 3. F.S. Prereq: 321, and 362 or 363, Engl 250. Survey of operating system issues. Introduction to hardware and software components including: processors, peripherals, interrupts, management of processes, threads and memory, deadlocks, file systems, protection, virtual machines and system organization, and introduction to distributed operating systems. Programming projects. Nonmajor graduate credit.

Textbook *Operating Systems Concepts: 6th Edition (2003)* by Silberschatz, Galvin, and Gagne, John Wiley & Sons

References

1. Modern Operating Systems: 2nd Edition by Andrew Tanenbaum, Prentice Hall, 2001.
2. Unix Network Programming: 2nd Edition by W. Richard Stevens, Prentice Hall, 1998.

Coordinators Dimitris Margaritis, Assistant Professor
Lu Ruan, Assistant Professor

Course Outcomes

- Students will learn about and understand theoretical concepts and programming constructs used for the operation of modern operating systems
- Students will gain practical experience with software tools available in modern operating systems such as semaphores, system calls, sockets and threads.

Relationship between Course Outcomes and Program Outcomes: C, D, H, K

Prerequisite by Topic

1. Computer Architecture
2. Programming in C++

Major Topics covered in the Course

1. System Structures Overview
2. Processes, threads, and CPU scheduling
3. Process synchronization and deadlocks
4. Memory Management and Virtual Memory
5. Distributed Systems Structures
6. File Systems

Laboratory projects

1. Creating concurrent processes using the fork system call and interprocess communication using pipes
2. Process synchronization using semaphores
3. Client/Server communication using sockets

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Students had problems understanding the concepts of semaphores and monitors.	Monitoring student grades in homework assigned on the topic.	Every semester	Students needed more instruction on the topic.	Instructor presented and discussed solutions to similar exercises in class in detail.
Students applying textbook synchronization solution to programming problem involving slightly different synchronization problem.	Conversations with students before project was due, in class.	Every semester	Theoretical concepts were not understood in depth.	Additional time in class was spent on the topics and more examples were presented.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

As indicated in the tables in Section 5-B, the credits for this course fall in the advanced category, because of the rigor and depth of the topics covered.

	CORE	ADVANCED		CORE	ADVANCED
Data Structures			Computer Organization and Architecture		1
Algorithms		1	Concepts of Programming Languages		
Software Design		1			

COURSE DESCRIPTION

Department and Course Com S 362 Course Instructor Les Miller
Number Fall 2008

Course Title Object-Oriented Analysis and Design Total Credits 3

Current Catalog Description (3-0) Cr. 3. F.S. Prereq: 228 with C- or better, Engl 250. Object-oriented requirements analysis and systems design. Design notations such as the Unified Modeling Language. Design Patterns. Group design and programming with large programming projects. Nonmajor graduate credit.

Textbook *Structure and Interpretation of Computer Programs, Second Edition* by [Harold Abelson](#) and [Gerald Jay Sussman](#) with [Julie Sussman](#) (MIT Press and McGraw-Hill, 1996).

Reference *The Little Schemer (Fourth Edition)* by [Daniel P. Friedman](#) and [Matthias Felleisen](#) (MIT Press, 1996).
The Java Programming Language (second edition) by Ken Arnold and James Gosling (Addison-Wesley, 1998).

Coordinators Leslie Miller, Professor

Course Outcomes

The main objective is that students will be able to analyze system requirements, and create and justify object-oriented designs that meet their requirements and that are robust and evolvable. In more detail, the essential outcomes for this course are that students will be able to:

- Analyze system requirements and model problem domains.
- Evaluate the quality of an analysis, and be able to explain how to improve it.
- Design and build object-oriented systems.
- Explain and justify designs based on design principles, patterns, and heuristics.
- Evaluate the quality of a design, and be able to explain how to improve it.
- Write object-oriented code to correctly implement a design.
- Be able to read and write analysis and design documentation in the Unified Modeling Language (UML).
- Be able to read and write object-oriented code, in Java, that uses subclasses, inheritance, abstract methods, subtypes, and subtype polymorphism.

Relationship between Course Outcomes and Program Outcomes: C, D, H, I, K

Prerequisites by Topic

1. Computer architecture and machine-level programming
2. Theory of computing
3. File organization and processing

Major Topics covered in the Course

1. Introduction
2. Procedural Abstraction
3. Data Abstraction
4. Modularity, Objects, State
5. Metalinguistic Abstraction
6. Course Summary and Evaluation

Laboratory projects

1. Using Scheme
2. Language Design and Recursion
3. Formulating Abstractions with Higher-order Procedures
4. Data Abstraction
5. Object-oriented techniques
6. Conventional interfaces and symbolic data
7. Java visitors and iterators, multiple representations, and systems with generic operations

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Evaluate the quality of a design, and be able to explain how to improve it.	Student performance on homework, team projects, and exams Feedback from companies and other courses	Every semester	Too much stress on completing the projects.	Moved core design material to the beginning of the semester.
Analyze system requirements and model problem domains.	Student feedback	Every semester	Confusion between system sequence diagrams and interaction diagrams.	Material on system sequence diagrams moved to later in the semester to allow students to more clearly understand the role of interaction diagrams.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	2	_____
Software Design	1	_____			

COURSE DESCRIPTION

Department and Course Com S 363 Course Instructor Shashi Gadia
Number Fall 2008

Course Title Introduction to Database Management Systems Total Credits 3

Current Catalog Description (3-0) Cr. 3. F.S. Prereq: 228 with C- or better, Engl 250. Relational, object-oriented, and semistructured data models and query languages. SQL, ODMG, and XML standards. Database design using entity-relationship model, data dependencies and object definition language. Application development in SQL-like languages and general purpose host languages with application warehouses, mediators and wrappers. Programming Projects. Non-major graduate credit.

Textbook Shashi K. Gadia. *An Elemental Introduction to Databases*. A set of class notes.

Coordinator Shashi K. Gadia, Associate Professor

Course Outcomes

- Students will learn about largeness of data and why it gives rise to stream-oriented processing and algebraic languages that are higher level than general-purpose programming language such as Java.
- Students will learn about storage and efficient retrieval of large information via algebraic query optimization and the use of indexing.
- Students will learn about multiple paradigms in databases: relational, object-oriented, and semistructured and appreciate why XML is a powerful emerging model for databases and its broader impact.
- Students will learn how data from multiple heterogeneous sources is integrated.

Relationship between Course outcomes and Program outcomes: H, I, J, K

Prerequisites by Topic

1. A High-level Programming Language
2. Introduction to Digital Design

Major Topics covered in the Course

1. Introduction:
Largeness of information and its implications on organization of disk in terms of blocks in kilobyte range rather than words consisting of a few bytes. Layered architecture of databases and varieties of users.
2. Query languages in different paradigms and hybrid languages
Foundational language relational algebra and how it gives rise to user-oriented language SQL. How SQL is reincarnated as OQL and XQuery in object-oriented and XML data models. Hybrid language JDBC (Java Database Connectivity).
3. Storage and efficient retrieval of data
Storing information in blocks on a disk. Indexing. Algebraic optimization. Query processing.
4. Schema design -
Schema design using Entity-relationship model and data
5. Relational, object-oriented, and semistructured (XML) data models
Programming projects.
6. Information integration: Data warehousing

Laboratory Projects

Projects on SQL, JDBC, Object-oriented query, and XQuery.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Understanding of largeness in information	Class discussions	Every semester	Give more details	- Through slower paced class discussion.
Broad impact of XML	Class discussions	Every semester	Give more details	- A lecture is exclusively devoted to XML to put it in broad perspective - Efficiency issues surrounding XML are discussed.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Algorithms	<u>0.50</u>	_____	Computer Organization and Architecture	<u>0.25</u>	_____
Data Structures	<u>0.25</u>	_____	Concepts of Programming Languages	<u>1.50</u>	_____
Software Design	<u>0.50</u>	_____			

COURSE DESCRIPTION

Department and Course Com S 409 Course Coordinator Carl Chang
Number Fall 2008

Course Title Software Requirements Engineering Total Credits 3

Current Catalog Description (Dual-listed with 509). (Cross-listed with S E). (3-0) Cr. 3. F. Prereq: Com S 309, Engl 250, Sp Cm 212. The requirements engineering process, including identification of stakeholders, requirements elicitation techniques such as interviews and prototyping, analysis fundamentals, requirements specification, and validation. Use of Models: State-oriented, Function-oriented, and Object-oriented. Documentation for Software Requirements. Informal, semi-formal, and formal representations. Structural, informational, and behavioral requirements. Use of requirements repositories to manage and track requirements through the life cycle. Case studies, software projects, written reports, and oral presentations will be required. Nonmajor graduate credit.

Textbook Mastering the Requirements Process (Suzanne & James Robertson, ISBN: 0-201-36046-2)

References

- Managing Software Requirements – A unified Approach (Dean Leffingwell and Don Widrig, ISBN: 0-201-61593-2)
- UML Toolkit by Hans-erik Eriksson and Magnus Penker.
- Writing effective use-cases, Alistair Cockburn, ISBN: 0-201-70225-8

Coordinator Carl Chang, Professor
Simanta Mitra, Senior Lecturer

Course Outcomes

1. Students will learn about the Requirements Engineering process.
2. Students will practice requirements analysis, modeling, elicitation, and specification on medium sized projects.
3. Students will learn to work with a team on requirements elicitation on medium sized projects and practice soft skills such as communication, presentation, and asking questions etc.
4. Students will become familiar with requirements management tools (such as Rational RequisitePro).
5. Students will be exposed to latest research in requirements engineering area.

Relationship between Course Outcomes and Program Outcomes: B, D, F, H, I, K

Prerequisite by Topics

1. Software Lifecycle
2. Development of Medium sized software projects
3. Proficiency in English

Major Topics covered in the Course

1. Requirements Analysis
2. Requirements Elicitation
3. Requirements Specifications
4. Requirements Validation and Verification
5. Requirements Management and Tools
6. Modeling and Modeling tools.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Learn about and practice requirements processes (Course Outcomes 1 and 2)	Student performance on screenshots, actors and use-cases, static and dynamic models of problem domain	Several times during the semester.	Hard to impress on students the difference in good requirements analysis and poor requirements analysis.	Multiple teams performed analysis for the SAME project. Later, findings of different teams are compared to drive home the point.
Team work (Course outcome 3)	Team performance in - reqs project - research project - textbook chapter presentations	Several times during semester.	Class has students of vastly different backgrounds with different levels of readiness to absorb information and learn.	Different levels of complexity for team projects. Novice students assigned project where TA and instructor are customers. Intermediate students assigned project where department staff and faculty are customers. Advanced students assigned project where external industry representatives are customers.
			Students tend to form teams with their friends or with similar nationality/ethnicity/background – thus diminishing lessons that can be learnt from team projects and increasing the differences with workplace teams.	Teams are imposed by the instructor with an eye towards making the teams diverse as possible. Forces students out of their comfort zones.
Learn soft and hard skills as needed for professional development and Use current techniques, skills and tools (Course outcomes 3 and 4)	Class Presentations, Project artifacts, Homeworks, Final interviews	Semester	Harder to teach soft skills. Text book is not really designed as an undergraduate text book.	Class organized around gathering “experiences” as opposed to lecture. Students learn soft skills by participation and engaging soft skills. Students work in teams to read and

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
			Could not find any suitable. Zero to no examples or questions at the end of chapters. Also – text book very easy to read – and lessons to be learnt from chapters can be missed.	present each chapter to the class. Discussion and question/answer session at the end of chapter. This approach allows more engagement with material than by pure lecture.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	_____
Software Design	_____	_____ 3			

COURSE DESCRIPTION

Department and Course Com S 417 Course Coordinator Simanta Mitra
Number Spring 2008

Course Title Software Testing Total Credits 3

Current Catalog Description (Cross-listed with S E). (3-0) Cr. 3. S. *Prereq: Com S 309, 319, Engl 250, Sp Cm 212.* Comprehensive study of software testing, principles, methodologies, management strategies and techniques. Test models, test design techniques (black box and white-box testing techniques), integration, regression, system testing methods, and software testing tools. Nonmajor graduate credit. Oral and written reports.

Textbook

Aditya Mathur, "Foundations of Software Testing", Addison-Wesley Professional, April 17, 2008.

References

1. Cem Kaner, Hung Nguyen, Jack Falk, "Testing Computer Software", 2nd Edition, John Wiley & Sons, April 1999. (ISBN: 0471358460).
2. Boris Beizer, "Black-Box Testing: Techniques for Functional Testing of Software and Systems", John Wiley & Sons, 1995. (ISBN: 0471120944)
3. Boris Beizer, "Software Testing Techniques", second edition, Vannostrand Reinhold, 1990.
4. Mark Fewster and Dorothy Graham, "Software Test Automation - Effective Use of Test Execution Tools", ACM Press and Addison-Wesley, 1999 (ISBN: 0-201-33140-3).
5. Elfriede Dustin, Jeff Rashka and John Paul, "Automated Software Testing: Introduction, Management, and Performance", Addison-Wesley Pub Co, 1999. (ISBN: 0201432870)

Coordinator Simanta Mitra, Senior Lecturer

Course Outcomes

At the end of the course, a student who passes is expected to have met the following learning objectives.

1. is knowledgeable about the basic vocabulary and concepts of this area, knows about the different test generation and selection techniques and is able to apply this knowledge.
2. understands the idea of software testability and designing for testability.
3. understands how inspections/reviews are conducted and why they are so important at all stages of the development process. Knows about the V-model.
4. has created and executed tests using testing tools (like JUnit/TestManager). Understands the need for automation and has used automation tools.
5. has designed and conducted testing on a software project starting from test plan to test report.
6. Recorded and managed information about defects found using a defect tracking tool. Debugged and fixed using IDE and managed different versions of software using source control tool.
7. Knows how to evaluate test cases and has evaluated test cases for the project using coverage tools.
8. is able to evaluate the software based on the tests.
9. knows about the issues and is able to apply knowledge of how to generate test cases for some domain-specific testing such as: GUI, OO, Web, Component Testing etc.
10. knows about the journals and conferences related to testing and has explored a topic and made a presentation to the class on this topic and written a report on this - has been exposed to contemporary work by researchers.

Relationship between Course Outcomes and Program Outcomes: C, D, F, H, I, K

Prerequisite by Topics

1. Software Lifecycle
2. Development of Medium sized software projects
3. Proficiency in English

Major Topics covered in the Course

1. Introduction to software testing
2. Black-box testing
3. White-box testing
4. Integration testing
5. Coverage and Regression testing
6. System testing and performance measurement
7. Software test automation and tools
8. Software quality assurance
9. Domain specific issues
 - a. Interprocedural
 - b. Concurrency
 - c. Object oriented
 - d. Performance
 - e. Web Based
 - f. GUI testing

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Students understand how to log bugs and then fix defects on a branch and have defects follow controlled bugs life cycle.(Course Outcome#7)	Subversion logs, bugzilla logs.	At the end of the testing project.	Earlier students had not been creating branches to fix bugs and instead were working on the trunk. The problem was figuring out how to create branches and tags.	Branching and Tagging were explained more rigorously and the testing project instructions were made more explicit.
Students need exposure to contemporary work by researchers (Course outcome#12)	Guest presenters, Research project outcomes.	At the end of the semester.	Need guest lecturers.	Asked department experts on testing to present. Dr. Samik Basu presented his work to the class.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	_____
Software Design	_____	_____ 3 _____			

COURSE DESCRIPTION

Department and Course Number Com S 418 Course Instructor Spring 2008 David Fernández-Baca

Course Title Introduction to Computational Geometry Total Credits 3

Current Catalog Description (Dual –listed with 518) (3-0) Cr. 3. Alt. S., offered 2005. Prereq: 311 or permission of instructor, Engl 105, Sp Cm 212. Introduction to data structures, algorithms, and analysis techniques for computational problems that involve geometry. Line segment intersection, polygon triangulation and visibility problems, range queries, point location, arrangements and duality, Voronoi diagrams and Delaunay triangulation, convex hulls. Other selected topics. Programming assignments. Nonmajor graduate credit.

Textbook M. de Berg et al. *Computational Geometry* (2nd edition). Springer-Verlag, 2000
ISBN: 3-540-65620-0.

Coordinator David Fernández-Baca, Professor

Course Outcomes: At the end of the course students are expected to have

- An ability to apply knowledge of computing and mathematics appropriate to the discipline.
- An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems that demonstrate comprehension of the tradeoffs involved in design choices.

Relationship between Course Outcomes and Program Outcomes: A, I, J

Prerequisite by Topic (s)

Major Topics covered in the Course

Our aim is to cover substantial portions of chapters 1 through 11 of the textbook. The topics are:

1. Introduction. Convex Hulls in two dimensions. Degeneracies and Robustness. Application Domains.
2. Line Segment Intersection. The Doubly-Connected Edge List. Computing the Overlay of Two Subdivisions. Boolean Operations
3. Guarding and Triangulations. Partitioning a Polygon into Monotone Pieces. Triangulating a Monotone Polygon.
4. Linear Programming. Half-Plane Intersection. Incremental Linear Programming. Randomized Linear Programming. Linear Programming in Higher Dimensions. Smallest Enclosing Discs.
5. Orthogonal Range Searching. Kd-Trees. Range Trees. Fractional Cascading.
6. Point Location and Trapezoidal Maps. A Randomized Incremental Algorithm. Dealing with Degenerate Cases.
7. Voronoi Diagrams. Definition and Basic Properties. Algorithms.
8. Arrangements and Duality. Computing the Discrepancy. Duality. Arrangements of Lines. Levels and Discrepancy.
9. Delaunay Triangulations. A Randomized Algorithm.

10. Additional Geometric Data Structures. Interval Trees. Priority Search Trees. Segment Trees.

11. Convex Hulls in 3-Space.

Laboratory projects: None

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
An ability to apply knowledge of computing and mathematics appropriate to the discipline.	Regular homework assignments and exams.	Several times a semester (every other year)	Students had difficulties applying concepts they had seen in earlier discrete math and algorithms classes to geometric problems.	More time was spent explaining the connection with previous courses.
An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems that demonstrate comprehension of the tradeoffs involved in design choices.	Regular homework assignments and exams.	Several times a semester (every other year)	Some students had a tendency to see only one “right” solution, when in reality there were a variety of solutions, which are appropriate for different settings, depending on whether speed of memory usage is an issue.	More examples were given in class. Additional exercises were assigned focusing on these concepts.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	2
Software Design	_____	1		_____	_____

COURSE DESCRIPTION

Department and Course Com S 430 Course Instructor Steve Kautz
Number Fall 2008

Course Title Advanced Programming Tools Total Credits 3

Current Catalog Description (3-1) Cr. 3. F. Prereq: 311, 362 or 363, Engl 250, Sp Cm 212. Topics in advanced programming techniques and tools widely used by industry (e.g., event-driven programming and graphical user interfaces, standard libraries, client/server architectures and techniques for distributed applications). Emphasis on programming projects in a modern integrated development environment. Oral and written reports. Nonmajor graduate credit

Textbook Brian Goetz et al, *Java Concurrency in Practice*, Addison-Wesley, 2006

References Patrick Niemeyer and Jonathan Knudsen, *Learning Java 3e*, O'Reilly, 2005
Gustavo Alonso et al, *Web Services: Concepts, Architectures, and Applications*, Springer, 2004
Gregor Hohpe and Bobby Wolfe, *Enterprise Integration Patterns*, Addison-Wesley, 2004
Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003

Coordinators Steve Kautz, Lecturer
James Lathrop, Senior Lecturer

Course Outcomes

At the end of the course the student should be able to:

- Write correctly synchronized multithreaded code; analyze safety and liveness properties of concurrent applications
- Design and implement an event-driven user interface of moderate complexity using a GUI framework
- Design and implement a web-based application in the form of services, with a user interface based on asynchronous messaging between browser and server
- Create a distributed system through the integration of heterogeneous applications and web services using appropriate tools and technologies
- Write requirements, API documentation, and end-user documentation for a system such as the above, and produce a professional oral presentation on it

Relationship between Course Outcomes and Program Outcomes: D, F, H, I, K

Prerequisite (s) by Topic

Design and analysis of algorithms, prior experience in a project-based computer science course, college-level written and oral communication skills.

Major Topics covered in the Course

- Concurrent programming; synchronization utilities; the Java memory model
- Design patterns for asynchronous activities
- Event-driven programming in the Swing GUI framework
- Swing models and the Model-View-Controller pattern
- HTTP and session state
- HTML-based user interfaces; JavaScript
- Asynchronous HTTP requests (Ajax)

- Issues in the integration of heterogeneous distributed systems; introduction to middleware
- Remote procedure calls and RMI; the Proxy pattern
- Introduction to CORBA and ICE (the Internet Communication Engine)
- Message queuing principles; introduction to MSMQ and JMS

Laboratory projects

1. Simple concurrent systems
2. A GUI frontend for a remote proxy
3. Proxy with concurrent requests
4. HTML-based frontend with asynchronous HTTP requests
5. Group project: integration of distributed legacy systems in a vehicle routing application

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Write correctly synchronized multithreaded code; analyze safety and liveness properties of concurrent applications	Exams, performance on programming projects.	Every year	Students have difficulty recognizing potential race conditions and understanding how to use synchronization utilities to resolve them	Create a sequence of exercises, starting early in the semester and gradually increasing in difficulty, emphasizing the use of synchronization utilities. Devote a portion of lecture time to case studies and common problems in multithreaded programming.
Write requirements, API documentation, and end-user documentation for a system and produce a professional oral presentation on it	Group project documentation and presentation	Every year	Students leave documentation to the end and produce minimal work of poor quality	Require first-draft written artifacts well before the project deadline; provide students with specific editorial feedback.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____	Concepts of Programming Languages	_____	<u>1</u>
Software Design	_____	<u>2</u>			

COURSE DESCRIPTION

Department and Course Com S 440 Course Instructor Hui-Hsien Chou
Number Spring 2009

Course Title Principle and Practices of Compiling Total Credits 3

Com S 440. Principles and Practice of Compiling. (Dual-listed with 540). (3-1) Cr. 3. Alt. S., offered 2009.
Prereq: 331, 342, Engl 250, Sp Cm 212. Theory of compiling and implementation issues of programming languages. Programming projects leading to the construction of a compiler. Projects with different difficulty levels will be given for 440 and 540. Topics: lexical, syntax and semantic analyses, syntax-directed translation, runtime environment and library support. Written reports. Nonmajor graduate credit.

Textbooks

1. Compilers: Principles, Techniques, and Tools (2nd Edition), by Aho, Lam, Sethi and Ullman. Addison-Wesley Publishing Company, 2006. ISBN 0-321-48681-1.
2. Programming for the Java Virtual Machine, by Joshua Engel. Addison-Wesley Publishing Company, 1999. ISBN 0-201-30972-6.

References

1. Lex & Yacc, by Levine, Mason and Brown. O'Reilly & Associates, 1992. ISBN 1-565-92000-7
2. The Java Virtual Machine Specification, 2nd Edition, by Lindholm and Yellin. Addison-Wesley Publishing Company, 1999. ISBN 0-201-43294-3. Also available from Sun's website.
3. The Java Language Specification, 3rd Edition (The Java Series), by Gosling, Joy, Steele, and Bracha, 2005. ISBN: 0-321-24678-0. Also available from Sun's website.
4. The Jikes Java compiler, <http://jikes.sourceforge.net/>.
5. Gnu Document Online, available from <http://www.gnu.org/manual/manual.html>. This Internet web site provides documents for many software tools which include Make, Flex, Bison, GCC, GDB, Gnu C Library, Emacs, among others.

Coordinator Hui-Hsien Chou, Associate Professor

Course Outcomes

Upon completion of this course, students will possess the following general skills and knowledge:

1. Thorough understanding of the overall architecture of a modern compiler.
2. Being familiar with both top-down and bottom-up parsing paradigms.
3. Fluent with syntax-directed translation scheme and different compiler-compilers.
4. Knowledgeable with assembly language and code-block based code generation scheme.
5. Knowing the inner details of compilers, libraries, operating systems/platforms, and how they interact with each other to form modern computing environments.
6. The experience and confidence of having developed a major software system with thousands of lines of code based on four years of CS training.

Relationship between Course Outcomes and Program Outcomes: A, C, F, H, I, J, K

Prerequisites by Topic

1. C or C++ programming experience. Also, knowledge of Java can help understand the Java Virtual machine, but is not required.
2. Theoretical ideas such as automata, languages, regular expressions and complexity issues.
3. Data structure ideas including trees, lists, symbol tables, hash tables, priority queues, etc.
4. Algorithm ideas including graph traversal, string searching and processing, parsing, sorting, etc.
5. Machine language and computer architecture experiences. This can be focused just on the Java Virtual Machine, as said above.
6. A previous exposure to a fundamental programming languages class.

Major Topics covered in the Course

1. Lexical analysis to recognize keywords in the source language (3 hours)
2. Symbol table to keep information gathered from the source language (3 hours)
3. Syntax analysis to understand the source language description (12 hours)
4. Semantic analysis to tell if the source language is meaningful (9 hours)
5. Code generation to turn the source language into machine codes (9 hours)
6. Code optimization to better translate the source language (3 hours)
7. Runtime libraries to support the execution of translated programs (3 hours)

Laboratory projects

1. A simple interpreter for a subset of the Postscript page description language (3 weeks)
2. A lexical analyzer for a subset of the Trend cellular automata programming language (2 weeks)
3. A parser for a subset of the Trend cellular automata programming language (3 weeks)
4. Provide runtime support for executing the compiled Trend programs (2 weeks)
5. Semantic tree construction for translating the Trend language into intermediates (3 weeks)
6. Code generation and integration with runtime environment for the Java Virtual Machine (3 weeks)

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Being familiar with both top-down and bottom-up parsing paradigms.	Projects and midterm exams	Every other year	Students tend to be familiar only with the parsing method they choose for their compiler; since most chose bottom-up parsing, they are not familiar with top-down parsing at all.	Added more material about top-down parsing in the lectures related to parsing strategy to balance the student preferences with important knowledge they should learn.
Fluent with syntax-directed translation scheme and different compiler-compilers.	Projects and midterm exams	Every other year	Some students do not have proper training in theoretical computer science and find it difficult to understand the principles of parser, parser-generators, and how they act on source code.	Added more examples in the class to show the mapping between a grammar, the parser table produced by the parser-generator, and the parsing action based on the table on actual source code.
Knowledgeable with assembly language and code-block based code generation scheme.	Projects and final exams	Every other year	Most students have not programmed in an assembly language before, and find it difficult to understand those machine instructions.	Lowered the expectation of student background in assembly language programming, and thoroughly introduced the Java Virtual Machine assembly language.

The experience and confidence of having developed a major software system with thousands of lines of code based on four years of CS training.	Projects and final exams	Every other year	Some students do not have experience in large-scale software development, so they end up failing the projects because they do not know to use make or to link their code against correct libraries, etc.	Asked the TA to provide more tutoring sessions to the students, and also introduced basic software building procedures and tools involved.
---	--------------------------	------------------	---	--

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	<u>1</u>	Concepts of Programming Languages	_____	<u>1</u>
Software Design	_____	<u>1</u>			

COURSE DESCRIPTION

Department and Course Com S 454 Course Instructor Wensheng Zhang
Number Spring 2009

Course Title Distributed and Network Operating Systems Total Credits 3

Current Catalog Description (Dual-listed with 554; same as Cpr E 454.) (3-1) Cr. 3. Alt. S., offered 2009. Prereq: 311, 352, Engl 105, Sp Cm 212. Laboratory course dealing with practical issues of design and implementation of distributed and network operating systems. These include task scheduling, storage management, resource arbitration and power management, threads, communication protocols, synchronization and concurrency control, and distributed middleware. Graduate credit requires additional in-depth study of these topics. Written reports. Nonmajor graduate credit.

Textbook None

Reference

1. Distributed Systems: Design and Paradigms, by Andrew S. Tanenbaum and Marrten van Steen.
2. TinyOS Tutorials (online)
3. TinyOS Enhancement Proposals (TEPs) (online)
4. TinyOS Programming, by Philip Levis (online)
5. Recent research papers on designing middleware atop TinyOS

Coordinators Johnny Wong, Professor
Wensheng Zhang, Assistant Professor

Course Outcomes

1. Students will learn the ability of using current techniques and tools to design and implement basic operating system components such as task scheduling, resource arbitration, communication, synchronization, and so on.
2. Students will learn to apply the design and development principles of distributed operating systems in the construction of distributed middleware components.
3. Students will learn to communicate effectively and learn to think creatively and critically, both independently and with others.

Relationship between Course Outcomes and Program Outcomes: C, F, H, I, K

Prerequisite(s) by Topic

1. Thorough Knowledge of C
2. Introduction to Operating Systems

Major Topics covered in the Course

1. Overview of Distributed and Network Operating Systems
2. nesC Language and Programming Techniques
3. TinyOS Execution Model and Task Schedulers
4. Communication Subsystem
5. Storage Management
6. Resource Arbitration and Power Management
7. Threads, Synchronization and Concurrency Control
8. Distributed Time Synchronization
9. Distributed Clustering, and Collaborative Scheduling
10. Distributed Data Management
11. Security Mechanisms: Data Encryption/Description, Key Management, Authentication and Integrity

Laboratory projects

1. Programming practice on nesC language and TinyOS execution model, and implementation of a new TinyOS scheduler
2. Implementation of a distributed shared storage system using TinyOS’ storage abstractions, resource arbitrations, and communication primitives.
3. Programming practice on multi-thread library, synchronization and concurrency control
4. (term project) Survey of a topic related to distributed and network operating systems, or design/implement a distributed middleware atop TinyOS.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Ability of using techniques and tools learned in OS courses to design and implement basic operating system components (task scheduling, resource arbitration, communication, synchronization, etc.) (G)	Student performance on laboratory projects, senior outcomes assessment surveys	Every other year	Students have rare chance to test their designed and implemented OS components on real computer systems.	1. Newly-emerging TinyOS/Sensor experimental tool is introduced to provide students with real and cutting-edge platforms for development and testing. 2. Selective real OS code is explained in class as examples illustrating live applications of OS design principle and techniques
Ability of applying design principles learned in OS courses to construct distributed middleware components of medium complexity (I)	Student performance on laboratory projects and term projects.	Every other year	Students often have difficulty for debugging distributed system components, especially in distributed embedded environment such as the TinyOS/sensor platform.	In-class discussion on how to use existing debugging tools effectively and efficiently, and how to develop new simple but effective debugging tools.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix B, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	1	Concepts of Programming Languages	_____	1
Software Design	_____	1			

COURSE DESCRIPTION

Department and Course Com S 455 Course Instructor Andrew Miner
Number Fall 2008

Course Title Simulation: Algorithms and Implementation Total Credits 3

Current Catalog Description (Dual-listed with 555.) (3-0) Cr. 3. F. Prereq: 311 and 330, Stat 330, Engl 150. Introduction to discrete-event simulation with a focus on computer science applications, including performance evaluation of networks and distributed systems. Overview of algorithms and data structures necessary to implement simulation software. Discrete and continuous stochastic models, random number generation, elementary statistics, simulation of queuing and inventory systems, Monte Carlo simulation, point and interval parameter estimation. Graduate credit requires additional in-depth study of concepts. Written reports. Nonmajor graduate credit

Textbook L. M. Leemis and S. K. Park, *Discrete-Event Simulation: A First Course*, Prentice Hall

Coordinator Andrew Miner, Associate Professor

Course Outcomes

Students will

1. Understand basics of discrete-event and Monte-Carlo simulations
2. Understand theory of random number generation and random variables
3. Understand theory and practice (algorithms) of collecting and computing statistics (for measurements generated by simulations)
4. Be able to construct computational models (simulations) of real-world discrete-event systems
5. Be able to identify current areas of research in discrete-event simulation

Relationship between Course Outcomes and Program Outcomes: A, B, F, H, I

Prerequisite by Topic

1. A high-level programming language
2. Calculus
3. Discrete math and probability

Major Topics covered in the Course

1. Modeling (by examples)
2. Random number generation
3. Monte Carlo simulation
4. Sample statistics
5. Next-event simulation
6. Discrete random variables (theory, and algorithms to generate them)
7. Continuous random variables (theory, and algorithms to generate them)
8. Output analysis (confidence intervals, batch means)

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
(2)	Student performance on homework and exams	Every year	Identified several sources of confusion for students (e.g., “Poisson” arrival process means “Exponential” interarrivals)	Lectures adjusted accordingly
(4)	Student performance on homework and exams; student comments on course evaluations	Every year	Relate simple textbook models to real-world systems	Motivating examples given in lecture and homework assignments
(4), (5)	Student comments	Every year	Some students would like to work on long-term projects	None yet, still working out details

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	2	Concepts of Programming Languages	_____	_____
Software Design	_____	1			

COURSE DESCRIPTION

Department and Course Com S 461 Course Instructor Ying Cai
Number Fall 2008

Course Title Introduction to Database Systems Total Credits 3

Current Catalog Description . (3-1) Cr. 3. F. Prereq: 311, Engl 250, Sp Cm 212 and Com S 363. Data models, database design theory, data storage, access methods, query evaluation and optimization, transaction management, concurrency control, spatial indexing, mobile database management, data mining, Internet search. Oral and written reports. Nonmajor graduate credit.

Textbook R. Ramakrishnan and J. Gehrke, Database Management Systems (3rd ed.), Mc Graw Hill

Reference Class notes on course web site <http://www.cs.iastate.edu/~cs461>
Silberschatz, Korth, Sudarshan, Database System Concepts, (5th ed.), Mc Graw Hill

Coordinators Shashi Gadia, Associate Professor
Leslie Miller, Professor
Ying Cai, Assistant Professor

Course Outcomes

- Students will learn about good database design techniques and database theories behind, including conceptual database designs, and functional dependencies and normalization
- Students will understand the design of important components of large-scale database management software, including query evaluation and optimization, transaction management, concurrency control.
- Student will understand a few advanced topics in database research, including spatial indexing, mobile objects management, data mining, and Internet search
- Student will experience working as a team and become familiar with some commercial relational database management system software

Relationship between Course Outcomes and Program Outcomes: D, F, H, I, K

Prerequisite by Topics

1. Data Structures and Algorithm Analysis
2. Introduction to Operating Systems

Major Topics covered in the Course

1. Why Database Systems
2. ER model and relational database design
3. Dependencies and normal forms
4. Storage management
5. Query evaluation and optimization
6. Relational algebra and calculus
7. Transaction management, recovery, and concurrency control
8. Spatial indexing
9. Moving object database management
10. Data mining

11. Internet search engine

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Comprehension of the major topics covered in the course	Each major topic is followed with a homework, which is graded by the course instructor. There are also midterm and final exams.	Every year	Students had difficulties in understanding dependency normalization and query evaluation	<ul style="list-style-type: none"> The class notes have been restructured More in-class examples are given
Experience working as a team and become familiar with some commercial relational database management system	Team software projects. Students were encouraged to talk to small business owners and other departments at ISU that needed database systems developed.	Every year	Students had difficulties in choosing an appropriate project that can be finished within one semester	<ul style="list-style-type: none"> Some projects done by earlier students are documents as examples

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____ 1 _____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____ 1 _____	Concepts of Programming Languages	_____	_____
Software Design	_____	_____ 1 _____			

COURSE DESCRIPTION

Department and Course Com S 472 Course Instructor Jin Tian
Number Fall 2008

Course Title Principles of Artificial Intelligence Total Credits 3

Current Catalog Description (Dual-listed with 572.) (3-1) Cr. 3. F. Prereq: 311, 330 or Cpr E 310, Stat 330, Engl 250, Sp Cm 212, Com S 342 or comparable programming experience. Specification, design, implementation, and selected applications of intelligent software agents and multi-agent systems. Computational models of intelligent behavior, including problem solving, knowledge representation, reasoning, planning, decision making, learning, perception, action, communication and interaction. Reactive, deliberative, rational, adaptive, learning and communicative agents and multiagent systems. Artificial intelligence programming. Graduate credit requires a research project and a written report. Oral and written reports. Nonmajor graduate credit

Textbook Artificial Intelligence: A Modern Approach (Second Edition) by Stuart Russell and Peter Norvig.

Reference <http://www.cs.iastate.edu/~cs472/>

Coordinators Vasant Honavar, Professor
Dimitris Margaritis, Assistant Professor
Jin Tian, Assistant Professor

Course Outcomes

- Appreciation of fundamental problems in artificial intelligence (AI).
- Ability to generate precise formulation(s) of AI problems in terms of knowledge representation and search from imprecise English description(s).
- Ability to design intelligent agents for problem solving, reasoning, planning, decision making, and learning.
- Ability to make intelligent choices from among available algorithms and knowledge representation schemes subject to specific design and performance constraints, and when needed, design variants of existing algorithms.
- Ability to implement and evaluate intelligent agents for representative AI problems – e.g., automated theorem proving, learning classification rules from data, etc.
- Familiarity with some current applications of AI.
- Ability to communicate effectively about AI problems, algorithms, implementations, and their experimental evaluation.

Relationship between Course Outcomes and Program Outcomes: A, B, D, F, I, J

Prerequisite by Topics

1. Knowledge of Data Structures (lists, trees, etc.)
2. Design and Analysis of Algorithms
3. Programming Language Concepts (e.g., functional programming, recursion, abstract data types) and ability to program in one or more modern object-oriented programming languages (e.g., JAVA).
4. Mathematics: Logic, Set Theory, Calculus, Probability Theory, Statistics
5. Working knowledge of modern programming environment(s), operating systems(s), WWW
6. Ability to write papers and reports and give presentations in English

Major Topics covered in the Course

- Overview of the course, overview of artificial intelligence, overview of intelligent agents

- Problem-solving and Search: Uninformed search, heuristic search, A*, local search, constraint satisfaction problem, games
- Knowledge Representation: Logical Agents, Propositional logic, First-Order Logic, Inference in First-Order Logic
- Uncertain Knowledge and Reasoning: Probability theory, Probabilistic Reasoning, Bayesian networks
- Learning Agents: Statistical learning, Bayesian decision theory, nearest neighbor classifiers, Naive Bayes model, Decision tree classifiers, Neural networks
- Selected Topics (As Time Permits): Planning Agents, Reasoning under assumptions, Reasoning about Knowledge, Customizing information retrieval, Automated knowledge discovery for diagnostics

Laboratory projects

- Problem Solving or Deliberative Agents
- Learning Agents
- Design of software agents (including problem solving agents, learning agents, and deliberative agents) for artificial intelligence applications, e.g., selective and customizable information retrieval, automated knowledge discovery. Independent project.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Having an understanding of the basic areas of AI including problem solving, knowledge representation, reasoning, and learning	Student performance in homework and exams	Annually	Students had problems formulating problems under a logical/AI framework in order to solve them using taught techniques.	-Expanded office hours for instructor so that students could have more frequent discussions. -Did more in-class problem-solving examples
Ability to design and implement intelligent agents for solving real problems	Project performance in solving problems using AI techniques	Annually	Students had trouble in deciding which technique to apply to a real problem.	Students prepared a one-page proposal of the problem and its solution. This was discussed in one-on-one meetings and improved upon before the final project work commenced.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	<u>1</u>	Concepts of Programming Languages	_____	<u>1</u>
Software Design	_____	<u>1</u>			

COURSE DESCRIPTION

Department and Course Com S 477 Course Instructor Yan-Bin Jia
Number Fall 2008

Course Title Problem Solving Techniques for Applied Total Credits 3
Computer Science

Current Catalog Description (Dual-listed with 577.) (3-0) Cr. 3. F. Prereq: 228, 330 or Cpr E 310, Math 166, Math 307 (or Math 317), or consent of the instructor. Selected topics in applied mathematics and modern heuristics that have applications in computer science and engineering. Perspective projection, homogeneous coordinates, rigid body rotations, parametric curves, curvature, torsion, surfaces, fundamental forms, Gaussian curvature, roots of polynomials, solution of linear and nonlinear equations, approximation, data fitting, fast Fourier transform, linear programming, nonlinear optimization, Lagrange multipliers, calculus of variations. Heavy programming components. Written report for graduate credit.

Textbook Course notes at <http://www.cs.iastate.edu/~cs577/>

Reference A number of reference books listed from the above website.

Coordinator Yan-Bin Jia, Associate Professor

Course Outcomes

- Students will learn some powerful techniques used in applications areas such as geometric modeling, graphics, robotics, vision, human machine interface, speech recognition, computer animation, etc.,
- Students will solve homework problems via implementation of some of the above techniques.
- Students are expected to achieve certain understanding of their applications in the real world.

Relationship between Course Outcomes and Program Outcomes: A, F, H, I, J

Prerequisite by Topics

1. Data Structures
2. Discrete Computational Structures
3. Calculus
4. Linear Algebra

Major Topics covered in the Course

1. Projections and homogeneous coordinates
 - a. Affine transformations
 - b. Homogeneous coordinates
 - c. Rotations & quaternions
2. Curves
 - a. Parametric curves
 - b. Curvature and torsion
 - c. Algebraic curves
3. Surfaces
 - a. Normal and geodesic curvatures
 - b. Principal curvature
 - c. Gaussian and mean curvatures

4. Roots of polynomials and nonlinear equations.
5. Solution of Linear Equations
 - a. LU decomposition
 - b. Singular value decomposition
6. Data fitting
 - a. Approximation by orthogonal polynomials
 - b. Fourier Transform & FFT
7. Nonlinear optimization
 - a. Conjugate gradient method
 - b. Lagrange multipliers
8. Calculus of variations
9. Linear programming & simplex algorithm

Laboratory projects

Programming assignments: 5 weeks

Programming projects on different scales are included in the assignments. The students are expected to implement a toolkit of techniques and apply them in problem solving. They will gain valuable experience in dealing with numerical issues in scientific computing in addition to developing software.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Comprehension of problem solving techniques	Student performance on homeworks and two exams which are graded by the instructor (and sometimes also by the TA if there is one).	Every year	Students tended to have difficulty with some topics. Students felt that some topics (such as resultants & numerical integration) were either not very useful or easy enough for self-study.	This has been addressed by giving more examples, and in the sample solution to every exercise problem. We have replaced some old topics for more interesting ones such as curves, surfaces, and variational techniques.
Applications of problem solving techniques	Student performance on homework projects.	Every year	Students had problem seeing how some methods are applied.	Talk about more examples on applications in the lecture, and include more "applied" homework problems and programming projects.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department's undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department's Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____1_____	Computer Organization and Architecture	_____	_____
Algorithms	_____	_____2_____	Concepts of Programming Languages	_____	_____
Software Design	_____	_____			

COURSE DESCRIPTION

Department and Course Com S 486 Course Instructor Johnny Wong
Number Spring 2009

Course Title Fundamental Concepts in Computer Total Credits 3
Networking

Current Catalog Description (3-0) Cr. 3. S. Prereq: 352. An introduction to fundamental concepts in the design and implementation of computer communication in both the wired and wireless networks, their protocols, and applications. Layered network architecture in the Internet, applications, transport, Socket APIs, network, and data link layers and their protocols, multimedia networking, and network security.

Textbook Computer Networking: A top-down approach featuring the Internet, 4th Edition, by James F. Kurose and Keith W. Ross, Addison Wesley, 2008.

Reference None

Coordinator Johnny Wong, Professor

Course Outcomes

- Students will learn to list and classify network services, protocols and architectures, explain why they are layered.
- Student will learn to explain key Internet applications and their protocols, and will apply to develop their own applications (e.g. Client Server applications, Web Services) using the sockets API.
- Students will learn to communicate effectively and learn to think creatively and critically through class programming projects both independently and collectively.

Relationship between Course Outcomes and Program Outcomes: D, F, H, I, K

Prerequisite by Topic(s)

1. Design and analysis algorithms
2. High-level programming languages of C, C++ or Java
3. Fundamental concepts in operating system

Major Topics covered in the Course

1. Computer networks and the Internet
2. Application layer
HTTP, DNS, and P2P applications
3. Transport layer
Reliable data transfer (GBN, SR) flow control and congestion control
4. Network layer
Routing algorithms, Internet addressing, Internet protocols
5. Link layer and local area networks
Multiple access protocols, link-layer addressing, Ethernet
6. Wireless and mobile networks
WiFi, mobile-IP, cellular networks
7. Network security
Authentication, Integrity, Availability, Encryption and Encryption with DES, RSA,

Laboratory projects

1. Programming practice of Client-Server application using C, C++ or Java
2. Programming projects with Web Services using sensors and/or Web Components Services in the Smart

Home Environment.

Assessment Plan for the Course

Outcome	How Measured	When Measured	Improvements Identified	Improvements Implemented
Comprehension of key internet applications and their protocols.	Performance on homework and exam	Every year	Students seemed to have diverse background of network programming.	Giving various examples of network programming in different languages. A useful reference or tutorial is available on the course website for student to understand network programming concept.
Comprehension of web services and applications	Performance on project	Every year	Students often have difficulty for debugging software components in Web Services applications	In-class discussion on how to use existing debugging tools effectively and efficiently, and how to develop new simple but effective debugging tools.

How Data in the Course is used to assess Program Outcomes

Assessment data from each course is given to the department’s undergraduate committee. A detailed description of how this data is used to assess program outcomes is provided in Appendix A, the department’s Learning Outcomes and Assessment Plan.

Curriculum Category Content (Semester Hours)

	CORE	ADVANCED		CORE	ADVANCED
Data Structures	_____	_____	Computer Organization and Architecture	_____	_____ 1
Algorithms	_____	_____ 1	Concepts of Programming Languages	_____	_____
Software Design	_____	_____ 1			