Computer Science Curriculum Review

Dr. Eric McGregor and Vincent Capaccio
January 2, 2015

Introduction

The ACM and IEEE Joint Task Force on Computing Curricula released their final report titled Computer Science Curricula 2013 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (CSC13)[1] on December 20, 2013. One of the main purposes of CSC13 was to provide institutions of higher education with a list of topics (a.k.a. student learning outcomes objectives) that the task force concluded should be included in an undergraduate curriculum. The CSC13 classifies these topics into three groups: Tier-1 Core, Tier-2 Core, and Elective. The task force defined 247 Tier-1 Core topics, 283 Tier-2 Core topics and 524 Elective topics and recommends that an undergraduate curriculum in computer science cover all Tier-1 Core topics, *a vast majority* of the Tier-2 Core topics and a *significant amount* of Elective topics.

In this curriculum review, we have analyzed the 2014-2015 Bridgewater College computer science curriculum and developed a set of proposals to accomplish the following.

1. To establish a curriculum that is near 100% compliant with CSC13 in support of President Bushman's goal of having curriculum compliant with outside standards.

2. To identify the topics that should be taught in our existing courses in order to provide our students with a comprehensive computer science education.

3. To update the department's Program Assessment Report SLOOs to reflect the changes in the curriculum.

Our primary goal was to establish a curriculum that is in line with CSC13. In order to do so, we considered all 247 Tier-1 and 283 Tier-2 topics provided in CSC13 as well as the Elective topics that pertain to the courses in our current curriculum. For each topic we determined which courses the topic should be taught in. Each topic was assigned to at least one course and some were assigned multiple courses.

During our analysis, we assigned the CSC13 topics to courses using a spreadsheet provided by the CSC13 task force. This spreadsheet lists all topics, one per row. We added additional columns to the spreadsheet, one column for each of the courses we offer. Then, for each topic, we placed 'X's in the columns that correspond to the courses where the topic will be taught. As we assigned topics to the existing courses we found that some topics did not fit into any of the existing courses. These included topics pertaining to "Software Patterns and Design" and "Computer Graphics". In order to ensure that all topics were assigned at least one course we created two additional columns for these two sets of topics.

After classifying the topics, we sorted the topics according to course and transferred the lists of topics to a text document, organized by course, and for each course, organized the topics according to subject matter and added headings. The list of topics by course can be found in Appendix A.

Below, in Section 1, we provide a list of the courses in the current curriculum along with the number of Tier-1 Core and Tier-2 Core topics that were identified as appropriate for our existing courses. In Section 2 we provide proposed changes to the curriculum along with justification for the suggested changes. Section 3 contains a graph of the current co/prerequisite scheme and of the co/pre-requisites that we believe are appropriate for the new curriculum. In Section 4 we provide recommendations on the time and frequency of the recommended course offerings. Section 5 contains a list of course descriptions that accurately reflect the topics that will be taught in the computer science courses and in Section 6 we establish a set of Program Assessment Report SLOOs. A conclusion is provided in Section 7. As mentioned earlier we provide in Appendix A a list each courses in the curriculum along with the topics that will be covered in each course.

Section 1. Current Computer Science Bachelors of Science Degree Requirements

The current Bachelor of Arts in Computer Science degree requires 47 credit hours to be completed. This includes 13 required courses (41 credits) and 2 elective courses (6 credits). There are 2 required MATH courses, 2 required CIS courses and 9 required CSCI courses. Students may choose their elective courses from a list containing 2 CIS courses, 2 ART courses, 3 MATH courses, 2 PHYS courses and 6 CSCI courses.

Below we list the courses for the current major along with the number of CSC13 Tier-1 and Tier-2 topics that were assigned to the courses during our analysis. During our analysis, we were able to assign 100% of the Tier-1 and Tier-2 topics in CSC13 to at least one course and some topics were assigned to multiple courses. The total topics column specifies the total number of Tier-1, Tier-2 and elective topics found in CSC13 for each course.

Our analysis identified a number of courses that should be removed from, added to, or modified in the major. These courses are highlighted below. A discussion of our proposed changes is found in Section 2.

Required Courses (41 credits)

Catalog #	Course Title	Tier-1	Tier-2	Total Topics	Credit Hours
MATH-131	Calculus I	0	0	0	3
MATH-132	Calculus II	0	0	0	3
CSCI-105	Introduction to Programming	21	4	25	4
CSCI-200	Intermediate Programming	32	12	47	4
CSCI-205	Data Structures	22	5	27	3
CSCI-225	Mathematical Structures	40	9	54	3
CSCI-315	Artificial Intelligence	4	6	10	3
CSCI-320	Algorithm Analysis	28	14	50	3
CSCI-340	Computer Architecture	8	34	42	3
CSCI-440	Operating Systems and Networking	36	66	113	3
CSCI-460	Seminar in Computer Science	0	0	0	3
CIS-250	Introduction to Information Systems	28	15	43	3
CIS-450	Software Engineering	17	37	54	3
	Total	236	200	436	41

Elective Courses (6 credits)

Catalog #	Course Name	Tier-1	Tier-2	Total Topics	Credit Hours
CSCI-140	Introduction to Web API Programming	0	1	6	3
CSCI-300	Software Practice	0	0	0	3
CSCI-330	Scripting Languages	0	0	0	3
CSCI-410	Signal/Image Processing	0	0	0	3
CSCI-430	Programming Paradigms	0	0	0	3
CSCI-435	Compiler Design	0	2	4	3
CIS-325	Data Communications	20	33	59	3
CIS-350	Database Management	6	22	62	3
ART-121	Introduction to Digital Media	0	0	0	3
ART-322	Web Theory and Design	0	0	0	3
MATH-310	Linear Algebra	0	0	0	3
MATH-341	Theoretical Statistics I	0	0	0	3
MATH-350	Numerical Analysis	0	0	0	3
PHYS-305	Electronics	0	0	0	3
PHYS-306	Digital Electronics	0	0	0	3
	Total	26	66	145	45

Topics Not Covered by Existing Courses

Category	Tier-1	Tier-2	Total Topics
Software Patterns and Design	14	19	34
Computer Graphics	4	5	9
Cryptography	0	6	12
Total	18	30	55

Section 2: Proposed Changes to the Computer Science Curriculum

In order to ensure that our students are exposed to nearly all of the recommended topics in CSC13 we propose a number of changes to the computer science curriculum. Our analysis shows that most of the recommended topics in CSC13 are covered by our existing required courses. However, a few issues need to be addresses.

1. It is unfeasible to cover all of the topics (113) that we've identified for CSCI-440 Operating Systems and Networking in a single course.
2. CSCI-460 Seminar in Computer Science is a required course, however, we did not identify any topics in CSC13 that are addressed in the course.
3. CIS-450 Software Engineering can not address a large number (34) of topics that CSC13 recommends due to the fact that most enrollees are ISM majors who have limited knowledge of software development.
4. CIS-350 Database Management is an elective course, however we've identified 28 Tier-1 and Tier-2 topics for this course.
5. There are 9 topics related to computer graphics in CSC13 and 12 for cryptography, yet we do not have a computer graphics or a cryptography course.
6. Mathematics, a field where students enhance their ability to express themselves mathematically, is important to the development of computer scientists. We would like to expand the options available to students in this area.
7. The current curriculum lists a number of electives that do not have a significant focus on computing.
8. We have new faculty with new interests that should be reflected in our electives.
9. With the reduction of general education requirements students are looking for additional elective courses. We seek to make available at least one elective each semester.

<u>Recommendations</u>

1. As mentioned above, it is unfeasible to cover the 113 topics we've identified for operating systems and networking. We therefore recommend splitting CSCI-440 Operating Systems and Networks into two courses: one for networks and security topics and another for operating systems topics. Specifically, we recommend changing the name of CSCI-440 to Networks and Security and creating a new required course, CSCI-330, called Operating Systems. CSCI-330 Operating Systems will cover 79 Tier-1 and Tier-2 topics. CSCI-440 Networks and Security will cover 76 Tier-1 and Tier-2 topics which include topics from CIS-325 (please see below).

2. Though there would an overlap in concepts covered in CSCI-440 and CIS-325, these courses would be taught in different manners. CSCI-440 would provide instruction and experience with writing client/server software, implementing network protocols and designing software for distributed systems, all of which are skills our computer science students should have and that ISM students are not prepared for and do not need for their careers. CIS-325, on the other hand, will expose our ISM students to "various standards, protocols, architectures, requirements, communication techniques and management issues". Since most of the topics covered in CIS-325 will be covered in CSCI-440 albeit in a different manner we propose to remove CIS-325 as an elective course for the computer science major.

3. Previously, CSCI-460 served as a vehicle for the senior comprehensive exam that was used for program assessment and as a capstone course where students developed some sort of software system. In Section 6 we propose a new system for assessment that does not include a comprehensive exam. In addition, since there is no course content, no lectures are given. Since we are no longer using the comprehensive exam for assessment we believe CSCI-460 should be replaced with another that covers some of the topics not covered in our existing courses and that includes a capstone project (See Recommendation 4). Therefore we propose to remove CSCI-460 as a course in our curriculum.

4. Most enrollees of CIS-450 Software Engineering are ISM majors. Understandably, these students have little experience developing software. As such, many topics recommended in CSC13 dealing with software engineering and all of the topics (34) on Software Patterns & Design can not be taught in CIS-450. In addition, though CIS-450 is named Software Engineering the course focuses on information systems design as described in the course description rather than software engineering. In the future we would like to make changes to the name and course description of CIS-450, but that is beyond the scope of our current analysis.

   We therefore propose to create a new course, CSCI-400 Software Engineering for Computer Scientists. The new course will cover all of the topics listed in Section 1 for CIS-450 as well as the 34 topics for Software Patterns & Design. This new course will also serve as a new capstone course for computer science majors requiring a capstone project. In conjunction we recommend removing CIS-450 as a required course for computer science majors. This will all us to change the focus of CIS-450 to information systems design and management which ISM majors can benefit from.

   Both CIS-450 and the new CSCI-400 will be taught to students in their final year at Bridgewater College. As such, enrollment will be based on the number of seniors in each program. We also expect additional enrollment in CSCI-400 by students who have taken CSCI-205, notably Physics majors taking the Physics and Technology track.

5. Currently CIS-350 Database Management is an elective course for computer science majors. Since it contains 28 topics recommended in CSC13, we propose to make CIS-350 a required course.

6. The authors of CSC13 claim that calculus courses are often required courses in computer science curriculum to help students achieve maturity in thinking and expressing themselves mathematically. The authors also state that other, equally strong, curriculum use Linear Algebra and Statistics to develop mathematic maturity in students. We agree with the authors and propose to remove Calculus I and II as required courses for the computer science major *and minor* and instead allow students to choose two math courses from the following list: Survey of Calculus or Calculus I, Calculus II, Introduction to Linear Algebra, Introduction to Statistics.

7. We believe computer science majors should choose their elective courses from a list of courses that deal with topics in computer science. We therefore propose to remove ART-121, MATH-310, MATH-341, and MATH-350 from the list of elective courses.

8. Since CSCI-315 Artificial Intelligence, has only 10 topics that have been identified as relevant to the course, we propose to move Artificial Intelligence from the set of required courses to the set of elective courses. We also recommend that we renumber CSCI-315 to CSCI-415 to reflect its rigor and so that if in the future CSCI-320 Algorithm Analysis is offered every year we might make CSCI-320 a prerequisite to Artificial Intelligence.

9. Because there are no topics assigned to CSCI-300 Software Practice, and our students receive a great deal of practice writing software in our curriculum, we propose to remove CSCI-300 Software Practice as a course in the curriculum.

10. Mobile computing is an increasingly relevant topic in computer science. We propose to add an elective course, CSCI-300, titled Mobile Application Development that will focus on the development of applications for mobile devices.

11. A number of topics (9) in CSC13 are devoted to computer graphics. We propose creating a new elective course, CSCI-305 named Computer Graphics.

12. Due to a lack of interest from students and the small number of CSC13 recommended topics (only 2) related to the course, we propose to remove CSCI-435 Compiler Design, from the courses we offer.

13. The term hacking refers to the act of penetrating a computer or network. Knowledge of the means by which hacking is performed is crucial to system administrators and software engineers. We therefore propose adding a new elective course CSCI-445 titled Ethical Hacking. This course will cover the ethics of hacking, hacking methods, and network penetration testing.

14. CSCI-430 Programming Paradigms, is an elective course taught during interterm. We have identified no CSC13 topics for this course and the faculty would prefer to offer other elective courses. Thus, we propose removing CSCI-430 from our course offerings.

15. CSCI-140 Introduction to Web API Programming Using Facebook, introduces students to the Javascript programming language and teaches them how to pull data from web services like Facebook and display it in a web page. As the course is an interterm course and it is impossible to teach the basics of programming as well as the techniques used to pull data from web services in 3 weeks, we propose making CSCI-105 a prerequisite for CSCI-140 and propose changing the course number of CSCI-140 to CSCI-240. We also propose to change the name of the course to *Web API Programming* and to make the course description more general to allow emphasis on other API, not just Facebook.

16. CSCI-330 Scripting Languages, a programming course, has only CSCI-105 as a prerequisite. Based on the catalog number, ISM students often wait till their third or fourth year to take CSCI-330, a required course for their major. This leaves them at a disadvantage since 2 or 3 years may have passed since they took CSCI-105, Introduction to Programming. We therefore propose changing the course number of CSCI-330 to CSCI-230 and will encourage advisors to advise students to take Scripting Languages in their second year.

17. Cryptography is an important and increasingly relevant field in computer science. We propose to add an elective course CSCI-420 Cryptography to the curriculum. This will be an interterm course that will introduce students to the algorithms and protocols used to encrypt data on computing devices and networks.

Our proposed changes will result in the following lists of courses for the computer science major.

Proposed Required Courses (35 credits)

Catalog #	Course Title	Tier-1	Tier-2	Total Topics	Credit Hours
CSCI-105	Introduction to Programming	21	4	25	4
CSCI-200	Intermediate Programming	32	12	47	4
CSCI-205	Data Structures	22	5	27	3
CSCI-225	Mathematical Structures	40	9	54	3
CSCI-320	Algorithm Analysis	28	14	50	3
CSCI-330	Operating Systems	23	56	90	3
CSCI-340	Computer Architecture	8	34	42	3
CSCI-400	Software Engineering for Comp. Sci.	31	56	88	3
CSCI-440	Networks and Security	33	43	82	3
CIS-250	Introduction to Information Systems	28	15	43	3
CIS-350	Database Management	6	22	62	3
	Topics	272	270	610	35

Proposed Required Math Courses (6 credits)

Catalog #	Course Title	Tier-1	Tier-2	Total Topics	Credit Hours
MATH-130	Survey of Calculus	0	0	0	3
MATH-131	Calculus I	0	0	0	3
MATH-132	Calculus II	0	0	0	3
MATH-140	Introduction to Statistics	0	0	0	3
MATH-210	Introduction to Linear Algebra	0	0	0	3
	Topics	0	0	0	15

Proposed Elective Courses (6 credits)

Catalog #	Course Name	Tier-1	Tier-2	Total Topics	Credit Hours
CSCI-230	Scripting Languages	0	0	0	3
CSCI-240	Introd. to Web API Programming	0	1	6	3
CSCI-305	Computer Graphics	4	5	9	3
CSCI-300	Mobile App Development	0	0	0	3
CSCI-410	Signal/Image Processing	0	0	0	3
CSCI-415	Artificial Intelligence	4	6	10	3
CSCI-420	Cryptography	0	6	12	3
CSCI-445	Ethical Hacking	0	0	0	3
ART-322	Web Theory and Design	0	0	0	3
PHYS-305	Electronics	0	0	0	3
PHYS-306	Digital Electronics	0	0	0	3
	Total	8	18	37	33

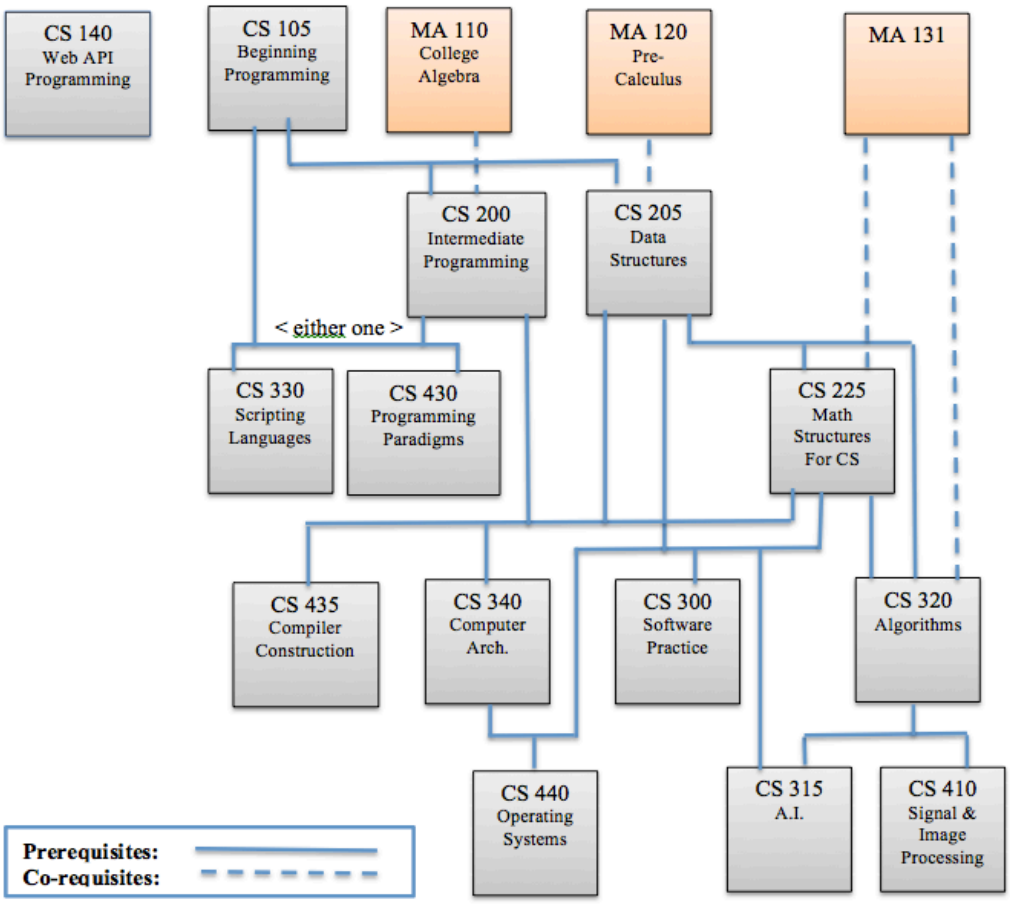
Our recommendations keep constant the number of required courses and electives for the major and keep constant at 47 the number of credit hours for the major. Our recommendations do however increase (+2) the number of CSCI courses provided by our department from 15 courses (9 required, 6 electives) to 17 (9 required, 8 electives). The additional 2 elective courses will be taught during the interterm.

We believe we will have sufficient enrollment for the new interterm courses. We have repeatedly heard from our computer science majors that they have taken the two interterm courses we offer and wish we would offer additional computer science courses during interterm. We also believe a number of students in the sciences will enroll, specifically students in the Mathematics and Physics programs.

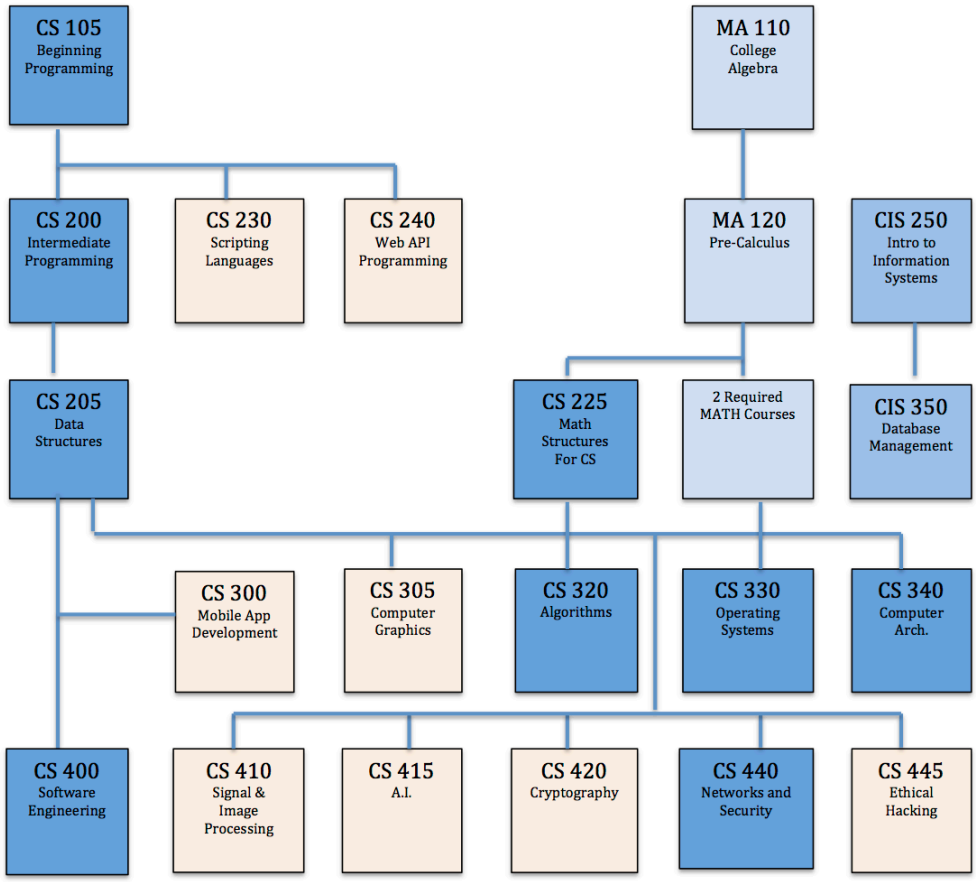
In addition, our proposal requires adding mathematics course options to the computer science minor and changing the course number for Scripting Languages in the ISM major.

Section 3: Modification to the Co-requisite and Prerequisite Designations

Current Computer Science Prerequisites and Co-requisites



Proposed Computer Science Prerequisites and Co-requisites



Here we summarize the changes to the computer science course pre-requisite and co-requisite designations. We use the new catalog course numbers that are proposed above.

1. CSCI-240 Web API Programming, teaches students how to use Javascript to interact with Internet services via service application programming interfaces (APIs) to produce content on a web page. In order to cover any significant material we must assume the student knows basic programming, otherwise the course is consumed with teaching the fundamentals of programming. Hence we recommend CSCI-105 Introduction to Programming as a prerequisite for CSCI-240.

2. We currently have math co-requisites for CSCI-200, CSCI-205 and CSCI-225. Our objective is to ensure students have taken the required mathematics courses before taking most 300 and 400 level computer science courses. The prerequisites below are simpler and will ensure our objective is met. Specifically, in Recommendation 7, we propose to require all required mathematics courses be taken before specific 300-400 level courses. We therefore recommend removing all mathematics co-requisites from CSCI-200, CSCI-205 and CSCI-225.

3. CSCI-205 investigates complex data structures that are implemented using advanced object-oriented concepts that are taught in CSCI-200. We recommend CSCI-200 Intermediate Programming as the only prerequisite for CSCI-205 Data Structures.

4. Since CSCI-225 builds upon concepts presented in MATH-120, Pre-Calculus we recommend MATH-120 as a prerequisite for CSCI-225.

5. We propose CSCI-205 Data Structures as a prerequisite to CSCI-300 Mobile Application Development.

6. We propose CSCI-205 Data Structures *and* permission by the instructor or senior standing in the computer science major as prerequisite to CSCI-400 Software Engineering for Computer Scientists. CSCI-400 will serve as a capstone course for computer science majors but we also want to allow other students to participate who have programming backgrounds and want to learn advanced software engineering techniques and participate in a large software engineering project.

7. We recommend CSCI-205 Data Structures, CSCI-225 Mathematical Structures in Computer Science and two mathematics courses from the list (MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210) as prerequisites for the following courses:

    - CSCI-305 Computer Graphics
    - CSCI-320 Algorithm Analysis
    - CSCI-330 Operating Systems
    - CSCI-340 Computer Architecture
    - CSCI-410 Signal and Image Processing
    - CSCI-415 Artificial Intelligence
    - CSCI-420 Cryptography
    - CSCI-440 Networks and Security
    - CSCI-445 Ethical Hacking.

8. Since all 300 and 400 level computer science courses are only offered every other year, it is pointless to require a 300 or 400 level course as a prerequisite to another 300 or 400 level course. We recommend removing all 300 level prerequisites for all 300 and 400 level courses. Note: When enrollment allows for 300 and 400 level courses to be taught on a yearly basis we may add additional prerequisites. The proposed course number scheme supports this.

Section 4: Proposed Changes to the Scheduling of Courses

Along with the proposed changes given above we recommend that the computer science courses be offered as follows. The highlighted rows identify course that we propose changes to when they are offered.

Catalog #	Course Title	Semester	Year	Credit Hours
CSCI-105	Introduction to Programming	F	Every	4
CSCI-200	Intermediate Programming	S	Every	4
CSCI-205	Data Structures	F	Every	3
CSCI-225	Mathematical Structures	S	Every	3
CSCI-320	Algorithm Analysis	F	Odd	3
CSCI-330	Operating Systems	S	Even	3
CSCI-340	Computer Architecture	F	Even	3
CSCI-400	Software Engineering For Comp. Sci.	S	Every	3
CSCI-440	Networks and Security	S	Odd	3
CIS-250	Introduction to Information Systems	S	Every	3
CIS-350	Database Management	F	Every	3
CSCI-230	Scripting Language	F	Even	3
CSCI-240	Introduction to Web API Programming	I	Even	3
CSCI-300	Mobile App Development	S	Even	3
CSCI-305	Computer Graphics	F	Odd	3
CSCI-410	Signal/Image Processing	S	Odd	3
CSCI-415	Artificial Intelligence	I	Even	3
CSCI-420	Cryptography	I	Odd	3
CSCI-445	Ethical Hacking	I	Odd	3

The changes we propose are summarized as follows.

1. We recommend that we offer CSCI-400 Software Engineering for Computer Scientists in the spring on a yearly basis to serve as a capstone course for seniors in the computer science major. CSCI-400 is a new course.

2. We recommend that CSCI-330 Operating Systems be offered in the spring of even years and that CSCI-440 Networks and Security, be offered in the spring of odd years. These courses replace CSCI-440 Operating Systems and Networking which is currently offered in the spring of even years.

3. We recommend offering CSCI-300 Mobile Application Development in the spring of even years. CSCI-300 is a new course.

4. We recommend offering CSCI-305 Computer Graphics in the fall of odd years. CSCI-305 is a new course.

5. We recommend offering CSCI-415 Artificial Intelligence in the interterm of even years. Currently Artificial Intelligence is a required course taught in the spring of odd years. Our proposal makes the course an elective thus suitable for an interterm course.

6. We recommend offering CSCI-420 Cryptography in the interterm of odd years. CSCI-420 is a new course.

7. We recommend offering CSCI-430 Ethical Hacking in the Interterm of odd years. CSCI-430 is a new course.

8. We recommend offering CSCI-240 Introduction to Web API programming in the interterm of even years. It is currently offered in the interterm of odd years.

Below we list course according to academic year (even and odd) along with the number of credit hours that will be taught each semester by CS faulty. CS faculty teaches courses shown in blue font and ISM faculty teaches courses shown in black font.

	Even Years (e.g. 2016-2017)	Hrs.	Odd Years (e.g. 2017- 2018)	Hrs.
F	CSCI-105 Intro. To Programming	8	CSCI-105 Intro. To Programming	8
	CSCI-205 Data Structures	3	CSCI-205 Data Structures	3
	CSCI-340 Computer Architecture	3	CSCI-320 Algorithm Analysis	3
	CIS-350 Database Management	*	CIS-350 Database Management	*
	CIS-450 Software Engineering	*	CIS-450 Software Engineering	*
	CSCI-230 Scripting Languages	3	CSCI-305 Computer Graphics	3
	Total CS Faculty Hours	17	Total CS Faculty Hours	17
I	CSCI-240 Into. To Web API Programming	3	CSCI-445 Ethical Hacking	3
	CSCI-415 Artificial Intelligence	3	CSCI-420 Cryptography	3
	Total CS Faculty Hours	6	Total CS Faculty Hours	6
S	CSCI-200 Inter. Programming	8	CSCI-200 Inter. Programming	8
	CSCI-225 Math Structures	3	CSCI-225 Math Structures	3
	CSCI-330 Operating Systems	3	CSCI-400 Software Engin. for Comp. Sci.	3
	CSCI-400 Software Engin. for Comp. Sci.	3	CSCI-440 Networks & Security	3
	CIS-250 Intro. To Information Sys.	*	CIS-250 Intro. To Information Sys.	*
	CSCI-300 Mobile App. Dev.	3	CSCI-410 Signal & Image Proc.	3
	Total CS Faculty Hours	16	Total CS Faculty Hours	16
	Total	39	Total	39

Our recommendations will impact department personnel as follows.

1. The Mathematics and Computer Science department has two Computer Science faculty members, a Visiting Professor and an Assistant Professor. Without considering overloads the computer science faculty are contracted to teach 45 credit hours per year. As shown above, our proposal includes 39 credit hours of instruction per year for the computer science major. This leaves 6 credit hours (or 12 if including overloads) per year available to the Computer Science faculty to teach CIS-350 and CIS-450 if needed, offer additional sections of CSCI-105 and CSCI-200 or to teach mathematics courses. No additional computer science faculty will be needed to implement the recommended changes.

   The computer science faculty are currently teaching three ISM courses as well as first year mathematics courses. Both the Mathematics and Computer Science and the Business departments are conducting searches to fill the ISM faculty position that was vacated by Dr. Lee Williams. If unsuccessful, our proposal will allow continued coverage of the ISM courses until a replacement is found. The Mathematics and Computer Science department is also planning on hiring a full-time lecturer and will not require the services of the computer science faculty to teach mathematics courses.

2. This proposal will require both faculty members to teach each year during the Interterm semester. Both CS faculty members have agreed to this measure.

A typical students 4-year schedule is shown below. Courses depicted in blue are required courses. Those in black are mathematics courses assumed chosen by the student. Elective courses are not shown. Our proposal ensures one elective is offered each semester.

Year	Sem.	Courses
2015	Fall	MATH-110 College Algebra CSCI-105 Introduction to Programming
2016	Spring	MATH-120 Pre-Calculus CSCI-200 Intermediate Programming CIS-250 Introduction to Information Systems
2016	Fall	MATH-131 Calculus I CSCI-205 Data Structures
2017	Spring	MATH-132 Calculus II CSCI-225 Math Structures in Computer Science
2017	Fall	CSCI-320 Algorithm Analysis CIS-350 Database Management
2018	Spring	CSCI-440 Networks & Security
2018	Fall	CSCI-340 Computer Architecture
2019	Spring	CSCI-330 Operating Systems CSCI-400 Software Engineering for Comp. Sci.

Year	Sem.	Courses
2016	Fall	MATH-110 College Algebra CSCI-105 Introduction to Programming
2017	Spring	MATH-120 Pre-Calculus CSCI-200 Intermediate Programming CIS-250 Introduction to Information Systems
2017	Fall	MATH-140 Introduction to Statistics CSCI-205 Data Structures
2018	Spring	MATH-210 Linear Algebra CSCI-225 Math Structures
2018	Fall	CSCI-340 Computer Architecture CIS-350 Database Management
2019	Spring	CSCI-330 Operating Systems
2019	Fall	CSCI-320 Algorithm Analysis
2020	Spring	CSCI-440 Networks & Security CSCI-400 Software Engineering for Comp. Sci.

Section 5: Course Catalog

Below we provide the course catalog for our proposed curriculum. Changes are shown in red font.

CSCI-105 Introduction to Programming
4 Credits F

This course is an introduction to computer programming using Java, a contemporary object-oriented language. Topics covered include the Java programming language and fundamental concepts for algorithm and software design. These include problem-solving methods, procedural and data abstraction, top-down modular design and proper programming style. Students gain experience using these skills to design, code, debug, and document computer programs. The course contains 3 credit hours of lecture and 2 hours of lab per week.

CSCI-200 Intermediate Programming
4 Credits S

This course will further develop and expand upon the topics introduced in CSCI-105. Topics will include object-oriented concepts such as inheritance, polymorphism and exception handling. Other topics include file I/O, multithreading and graphical user interfaces. The object-oriented programing language Java will be used to illustrate these topics. Problem solving, algorithm development, program design, and testing are emphasized. The course contains 3 credit hours of lecture and 2 hours of lab per week.

Prerequisites: CSCI-105

CSCI-205 Data Structures
3 Credits F

Advanced programming techniques will be covered with extensive use of recursion and dynamic data structures. Abstract data types including lists, stacks, queues, trees and hash tables are studied. Algorithms for searching and sorting are explored. The topics in this course provide an essential foundation for the further study of computer science. The object-oriented programing language Java will be used to illustrate these topics.

Prerequisites: CSCI-200

CSCI-225 Mathematical Structures for Computer Science
3 Credits S

This course is an introduction to the fundamental mathematical concepts and structures used in computer science. Topics include propositional and predicate logic; sets, functions and relations; mathematical induction, counting principles and recurrences; trees and graphs. Topics implemented in C++, Java, Ruby, Python, or mathematical programming languages such as Mathematica or MATLAB.

Prerequisites: MATH-120

CSCI-230 Scripting Languages
3 Credits F

Scripting languages are regularly used in server environments to automate tasks. This course introduces the student to the Linux operating system, which is often used in enterprise servers. Students will learn how to write scripts to automate tasks using the Unix shell and other scripting languages such as Perl, Python, and Ruby. Regular expressions and their use with common Unix commands such as grep, sed and awk/gawk are discussed. Process control, file systems, redirection, pipes, and scheduling tasks with cron are also discussed.

Prerequisites: CSCI-105
Alternate years: offered 2016–2017

CSCI-240 Web API Programming
3 Credits I

This course is an introduction to using HTML, Javascript, JQuery and web application programming interfaces (APIs) for web services like Facebook and Google Maps. Upon completion, the student will be able to create web sites with dynamic content using JQuery widgets, plugins, and data pulled from other web services.

Prerequisites: CSCI-105
Alternate years: offered 2016-2017

CSCI-300 Mobile Application Development
3 Credits S

This course is an introduction to Android application development. Students will learn the core skills and practices used to develop and test Android applications. Topics include the Android activity life cycle, user interface components and layouts, data storage, messaging, and content providers. Students are required to purchase a test Android phone for this course.

Prerequisites: CSCI-205
Alternate years: offered 2016-2017

CSCI-305 Computer Graphics
3 Credits F

This course is an introduction to computer graphics. Graphics hardware, rendering APIs and algorithms for displaying 2D and 3D objects in animations and interactive displays are discussed. The OpenGL and WebGL APIs are used for demonstrations and exercises.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2015-2016

CSCI-320 Algorithm Analysis
3 Credits F

This course analyzes the impact of data structure design on algorithm design and performance. Topics include graph and tree algorithms, performance analysis, testing and classification of algorithms, and design techniques. Topics implemented in C++, Java, Ruby, Python, or mathematical programming languages such as Mathematica

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2015–2016

CSCI-330 Operating Systems
3 Credits S

This course covers principles of computer operating systems including the management of processes, memory, I/O devices, and file systems. Other topics include issues of scheduling, security, and concurrency, distributed systems and virtualization. Students will gain practical experience working with the LINUX operating system and the C programming language.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210.
Alternate years: offered 2016–2017

CSCI-340 Computer Architecture
3 Credits F

This course is an introduction to computer systems and their organization. Topics include logic, gates, components and system level organization of generic computing systems. Bus architecture, memory organization, data representation and processor design are discussed. Includes an introduction to assembly language programming with appropriate laboratory assignments.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2016–2017

CSCI-400 Software Engineering for Computer Scientists
3 Credits S

This course is a project-based course that covers the tools and processes used in modern software development. Students will work as a team to design, implement, test and document a software system for an outside client. Lecture topics include software development strategies, problem elicitation, analysis and modeling, patterns, and team communication.

Prerequisites: Senior standing in the computer science major or both CSCI-205 and permission by instructor.

CSCI-410 Signal and Image Processing
3 Credits S

Signal and image processing are studied using modern signal and image processing function libraries to explore and program waveform analysis, convolution and correlation including FIR filters, spectrum analysis and composing linear systems. Image processing expands FIR filtering from one dimension to two dimensions and studies applications such as image contouring, edge detection, smoothing and noise removal. Programming is required using signal and image processing APIs.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2015–2016

CSCI-415 Artificial Intelligence
3 Credits I

This course is an introduction to the field of Artificial Intelligence (AI). Discusses what AI is, surveys some of the major results in the field and looks at a few promising directions. Covers AI problem-solving, knowledge representation, reasoning, planning and machine learning in detail with exercises that expose students to various AI systems and languages. Advanced topics such as natural language processing, vision, robotics and uncertainty are also covered at a survey level. Topics implemented in C++, Java, Ruby, Python, or mathematical programming languages such as Mathematica or MATLAB.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2016–2017

CSCI-420 Cryptography
3 Credits I

This course is an introduction to cryptography. The course will present terminology, principles, algorithms and tools related to cryptography and cryptanalysis including public and symmetric key cryptosystems, key exchange, authentication protocols and digital signatures.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2015-2016

CSCI-440 Networks and Security
3 Credits S

This course covers network organization and layered networking protocols including common Internet protocols such as TCP, IP and SMTP. Other topics include wireless communications, distributed programming, network security and client/server programming. Topics implemented in C++, Java, Ruby or Python.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2015–2016

CSCI-445 Ethical Hacking
3 Credits I

This course covers methods attackers use to target networks, the tools attackers use, and how these methods can be used by ethical hackers to discover weaknesses in a network with the ultimate goal of securing a network. Topics include footprinting, attack vectors, and intrusion detection systems.

Prerequisites: CSCI-205, CSCI-225 and two math courses from the following list: MATH-130 or MATH-131, MATH-132, MATH-140, MATH-210
Alternate years: offered 2015–2016

Section 6: Program Assessment Report SLOOs

Following the development of the proposed Computer Science curriculum we've established 9 Program Assessment Report SLOOs for our annual assessment. These SLOOs were developed using the list of topics from CSC13 that we plan to teach in our courses. Each of the SLOOs will be evaluated during the course final exam.

CSCI-200 Intermediate Programming

Students can …

1. Use subclassing to design simple class hierarchies that allow code to be reused for distinct subclasses.
2. Write a program that uses file I/O to provide persistence across multiple executions.
3. Demonstrate the creation and use of an interfaces.
4. Build robust code using exception handling mechanisms.

CSCI-205 Data Structures

Students can …

1. Write programs that use the following data structures: arrays, linked lists, stacks, queues, sets, and maps.
2. Demonstrate different traversal methods for trees and graphs, including pre, post, and in-order traversal of trees.
3. Utilize an iterator object from a collection class.
4. Implement a simple recursive functions and procedures.

CSCI-225 Mathematical Structures in Computer Science

Students can …

1. Design a deterministic finite state machine to accept a specified language.
2. Use the rules of inference to construct proofs in propositional and predicate logic.
3. Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application.
4. Calculate probabilities of events and expectations of random variables for elementary problems such as games of chance.

CSCI-320 Algorithm Analysis

Students can …

1. Use big O notation formally to give asymptotic upper bounds on time and space complexity of algorithms.
2. Demonstrate the difference between brute force, greedy, divide and conquer, and dynamic backtracking algorithms.
3. Implement simple search algorithms and explain the differences in their time complexities

CSCI-340 Computer Architecture

Students can …

1. Explain the organization of the classical von Neumann machine and its major functional units.
2. Articulate that there are many equivalent representations of computer functionality, including logical expressions and gates, and be able to use mathematical expressions to describe the functions of simple combinational and sequential circuits
3. Write simple assembly language program segments.
4. Explain the basic concepts of interrupts and I/O operations

CSCI-440 Operating Systems

Students can …

1. Demonstrate using a high-level programming language how to prevent a race condition from occurring and how to handle an exception.
2. Implement simple schedule algorithms.
3. Summarize the principles of virtual memory as applied to caching and paging.
4. Demonstrate on an execution time line that parallelism events and operations can take place simultaneously (i.e., at the same time). Explain how work can be performed in less elapsed time if this can be exploited.

CSCI-400 Software Engineering for Computer Scientists

Students can …

1. Identify and justify necessary roles in a software development team.
2. Differentiate among the phases of software development.
3. Construct models of the design of a simple software system that are appropriate for the paradigm used to design it.
4. Create a unit test plan for a code segment.

CSCI-325 Networks and Security

Students can …

1. Define the principles behind naming schemes and resource location
2. Compare common network organizations, such as Ethernet/bus, ring, switched vs. routed.
3. Describe the operation of reliable delivery protocols.
4. Compare and contrast weakest-link vs. defense in depth.

CIS-350 Database Management

Students can …

1. Develop and EER diagram given a problem description.
2. Prepare a relational schema from a conceptual model developed using the entity-relationship model
3. Create a relational database schema in SQL that incorporates key, entity integrity, and referential integrity constraints.
4. Use a declarative query language to elicit information from a database.

Section 7: Conclusion

In December 2013, the ACM and IEEE Joint Task Force on Computing Curricula released a guide (CSC13) for undergraduate curriculum in computer science. The guide contains 530 topics that they recommend be taught in a computer science curriculum. Using their recommendations we've reviewed the Bridgewater College computer science major and proposed a curriculum that is nearly 100% compliant with their recommendations.

In general our proposal recommends

1. Removing from the list of required courses, courses that do not teach a significant number of topics given in CSC13
2. Making required those courses that are currently electives but which have a significant number of topics given in CSC13
3. Adding additional electives to provide our students with ample opportunity to be exposed to a wide range of topics in the computer science field.

The proposal leaves unchanged the number of credits required for the major and does not require additional faculty to implement. We do however propose to increase the number of elective courses by 2 to provide more opportunities to our students. We believe the changes we've proposed will enhance the students experience at Bridgewater College and will provide them with the necessary skills to demonstrate their ability to graduate and qualify for jobs in their fields.

Appendix A

CSCI 105 - Introduction to Programming

Ethics

1. Evaluate the professional codes of ethics from the ACM, the IEEE Computer Society, and other organizations.

Modeling & Algorithms

2. Explain the concept of modeling and the use of abstraction that allows the use of a machine to solve a problem.
3. Discuss the importance of algorithms in the problem-solving process.
4. Discuss how multiple algorithms, each with different properties, can solve a problem.
5. Create algorithms for solving simple problems.
6. Apply the techniques of decomposition to break a program into smaller pieces.
7. Analyze simple problem statements to identify relevant information and select appropriate processing to solve the problem.

Programming Language Constructs

8. Construct and debug programs using the standard libraries available with a chosen programming language.
9. Understand common classes of input validation errors, and be able to write correct input validation code
10. Discuss the appropriate use of built-in data structures.
11. Identify and describe uses of primitive data types.
12. Write programs that use primitive data types.
13. Modify and expand short programs that use standard conditional and iterative control structures and functions.
14. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, the definition of functions, and parameter passing.
15. Choose appropriate conditional and iteration constructs for a given programming task.
16. Perform computations involving modular arithmetic.
17. Explain why everything is data, including instructions, in computers
18. Explain the reasons for using alternative formats to represent numerical data
19. Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays

Implementation

20. Implement basic numerical algorithms
21. Use a programming language to implement, test, and debug algorithms for solving simple problems.
22. Trace the execution of a variety of code segments and write summaries of their computations.
23. Construct, execute and debug programs using a modern IDE and associated tools such as unit testing tools and visual debuggers.
24. Analyze the extent to which another programmer's code meets documentation and programming style standards.
25. Apply consistent documentation and program style standards that contribute to the readability and maintainability of software.

CSCI 200 - Intermediate Programming

Programming Language Constructs

1. Explain why you might choose to develop a program in a type-safe language like Java, in contrast to an unsafe programming language like C/CZZ
2. Discuss the appropriate use of built-in data structures.
3. Explain benefits and limitations of static typing.
4. Write basic algorithms that avoid assigning to mutable state or considering reference equality.
5. Write useful functions that take and return other functions.
6. Construct and debug programs using the standard libraries available with a chosen programming language.
7. Write a program that uses file I/O to provide persistence across multiple executions.

8. Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit.

Object Oriented Constructs

9. Compare and contrast (1) the procedural/functional approach—defining a function for each operation with the function body providing a case for each data variant—and (2) the object-oriented approach—defining a class for each data variant with the class definition providing a method for each operation. Understand both as defining a matrix of operations and variants.
10. Use subclassing to design simple class hierarchies that allow code to be reused for distinct subclasses.
11. Correctly reason about control flow in a program using dynamic dispatch
12. Identify the data components and behaviors of multiple abstract data types.
13. Implement a coherent abstract data type, with loose coupling between components and behaviors.
14. Describe the main concepts of the OO model such as object identity, type constructors, encapsulation, inheritance, polymorphism, and versioning.
15. Define and use program pieces (such as functions, classes, methods) that use generic types.
16. Use multiple encapsulation mechanisms, such as function closures, object-oriented interfaces, and support for abstract datatypes, in multiple programming languages.
17. Explain the relationship between object-oriented inheritance (code-sharing and overriding) and subtyping (the idea of a subtype being usable in a context that expects the supertype).

Software Design

18. Explain the concept of modeling and the use of abstraction that allows the use of a machine to solve a problem.
19. Apply the techniques of decomposition to break a program into smaller pieces.
20. Create algorithms for solving simple problems.
21. Explain why the creation of correct program components is important in the production of high-quality software.
22. Articulate design principles including separation of concerns, information hiding, coupling and cohesion, and encapsulation.
23. Discuss the advantages and disadvantages of software reuse.
24. Analyze simple problem statements to identify relevant information and select appropriate processing to solve the problem.

Secure, Defensive and Robust Programming

25. Understand that an adversary controls the input channel and understand the importance of input validation and data sanitization.
26. Understand common classes of input validation errors, and be able to write correct input validation code
27. Demonstrate the identification and graceful handling of error conditions.
28. Describe how a contract can be used to specify the behavior of a program component.
29. Identify common coding errors that lead to insecure programs (e.g., buffer overflows, memory leaks, malicious code) and apply strategies for avoiding such errors.
30. Describe secure coding and defensive coding practices.
31. Explain the distinction between program errors, system errors, and hardware faults (e.g., bad memory) and exceptions (e.g., attempt to divide by zero).
32. Articulate the distinction between detecting, handling, and recovering from faults, and the methods for their implementation.
33. Build robust code using exception handling mechanisms.
34. For multiple programming languages, identify program properties checked statically and program properties checked dynamically. Use this knowledge when writing and debugging programs.
35. Describe software development best practices for minimizing vulnerabilities in programming code.

Implementation, Debugging & Testing

36. Implement basic numerical algorithms
37. Create a unit test plan for a medium-size code segment.
38. Use a programming language to implement, test, and debug algorithms for solving simple problems.
39. Construct, execute and debug programs using a modern IDE and associated tools such as unit testing tools and visual debuggers.
40. Analyze the extent to which another programmer's code meets documentation and programming style standards.
41. Apply consistent documentation and program style standards that contribute to the readability and maintainability of software.
42. Conduct a personal code review (focused on common coding errors) on a program component using a provided checklist.
43. Contribute to a small-team code review focused on component correctness.
44. Refactor a program by identifying opportunities to apply procedural abstraction.

45. Apply a variety of strategies to the testing and debugging of simple programs.

46. Describe techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness.
47. Use static and dynamic tools to identify programming faults

CSCI 205 - Data Structures

General Programming

1. Understand that an adversary controls the input channel and understand the importance of input validation and data sanitization
2. Understand common classes of input validation errors, and be able to write correct input validation code
3. Demonstrate the identification and graceful handling of error conditions.
4. Write a program that uses file I/O to provide persistence across multiple executions.
5. Identify the relative strengths and weaknesses among multiple designs or implementations for a problem.

Data Structures

6. Discuss the appropriate use of built-in data structures.
7. Describe common applications for each data structure in the topic list.
8. Write programs that use each of the following data structures: arrays, strings, linked lists, stacks, queues, sets, and maps.
9. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
10. Choose the appropriate data structure for modeling a given problem.
11. Understand the implementation of hash tables, including collision avoidance and resolution.
12. Compare alternative implementations of data structures with respect to performance.
13. Understand the heap property and the use of heaps as an implementation of priority queues.

Graphs & Trees

14. Solve problems using fundamental graph algorithms, including depth-first and breadth-first search.
15. Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each type of graph/tree.
16. Demonstrate different traversal methods for trees and graphs, including pre, post, and in-order traversal of trees.
17. Model a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology or the organizaiton of a hierarchical file system.
18. Show how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting.
19. Solve problems using graph algorithms, including single-source and all-pairs shortest paths, and at least one minimum spanning tree algorithm
20. Explain how to construct a spanning tree of a graph.
21. Determine if two graphs are isomorphic.

Iterators

Define and use iterators and other operations on aggregates using idioms most natural in multiple programming languages, including taking functions as arguments.

Recursion

22. Describe the concept of recursion and give examples of its use.
23. Implement, test, and debug simple recursive functions and procedures.
24. Identify the base case and the general case of a recursively-defined problem.
25. Determine whether a recursive or iterative solution is most appropriate for a problem
26. Use recursive backtracking to solve a problem such as navigating a maze

CSCI 225 – Mathematical Structures in Computer Science

Theory of Computation

1. Discuss the concept of finite state machines.
2. Design a deterministic finite state machine to accept a specified language.
3. Create state and transition diagrams for simple problem domains
4. Explain why the halting problem has no algorithmic solution.
5. Describe computations as a system with a known set of configurations, and a byproduct of the computation is to transition from one unique configuration (state) to another (state).

6. Recognize the distinction between systems whose output is only a function of their input (Combinational) and those with memory/history (Sequential).
7. Describe a computer as a state machine that interprets machine instructions.
8. Explain how a program or network protocol can also be expressed as a state machine, and that alternative representations for the same computation can exist.
9. Develop state machine descriptions for simple problem statement solutions (e.g., traffic light sequencing, pattern recognizers).
10. Derive time-series behavior of a state machine from its state machine representation.
11. Generate a regular expression to represent a specified language.
12. Design a context-free grammar to represent a specified language.
13. Distinguish syntax and parsing from semantics and evaluation.
14. Use formal grammars to specify the syntax of languages.
15. Explain the Church-Turing thesis and its significance.
16. Explain Rice's Theorem and its significance.
17. Provide examples of un-computable functions.

Symbolic Logic

18. Convert logical statements from informal language to propositional and predicate logic expressions.
19. Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms.
20. Use the rules of inference to construct proofs in propositional and predicate logic.
21. Describe how symbolic logic can be used to model real-life situations or applications, including those arising in computing contexts such as software analysis (e.g., program correctness), database queries, and algorithms.
22. Describe the strengths and limitations of propositional and predicate logic.
23. Apply the pigeonhole principle in the context of a formal proof.
24. Apply formal logic proofs and/or informal, but rigorous, logical reasoning to real problems, such as predicting the behavior of software or solving problems such as puzzles.
25. Translate a natural language (e.g., English) sentence into predicate logic statement.
26. Convert a quantified logic statement into clause form.
27. Apply resolution to a set of logic statements to answer a query.

Proof Techniques

28. Identify the proof technique used in a given proof.
29. Outline the basic structure of each proof technique described in this unit.
30. Apply each of the proof techniques correctly in the construction of a sound argument.
31. Determine which type of proof is best for a given problem.
32. Explain the relationship between weak and strong induction and give examples of the appropriate use of each.
33. Analyze basic logical fallacies in an argument.
34. Analyze an argument to identify premises and conclusion.
35. State the well-ordering principle and its relationship to mathematical induction.

Counting

36. Explain with examples the basic terminology of functions, relations, and sets.
37. Perform the operations associated with sets, functions, and relations.
38. Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.
39. Apply counting arguments, including sum and product rules, inclusion-exclusing principle and arithmetic/geometric progressions.
40. Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application.
41. Map real-world applications to appropriate counting formalisms, such as determining the number of ways to arrange people around a table, subject to constraints on the seating arrangement, or the number of ways to determine certain hands in cards (e.g., a full house).

Probability & Statistics

42. Calculate probabilities of events and expectations of random variables for elementary problems such as games of chance.
43. Differentiate between dependent and independent events.
44. Identify a case of the binomial distribution and compute a probability using that distribution.
45. Make a probabilistic inference in a real-world problem using Bayes' theorem to determine the probability of a hypothesis given evidence.
46. Compute the variance for a given probability distribution.

47. Explain how events that are independent can be conditionally dependent (and vice-versa). Identify real-world examples of such cases.
48. Apply Bayes theorem to determine conditional probabilities in a problem.
49. Apply Bayes' rule to determine the probability of a hypothesis given evidence.

Trees

50. Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each type of graph/tree.
51. Show how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting.

Recursion

52. Describe the concept of recursion and give examples of its use.
53. Identify the base case and the general case of a recursively-defined problem.
54. Explain the parallels between ideas of mathematical and/or structural induction to recursion and recursively defined structures.

CSCI 315 – Artificial Intelligence

Simulation

1. Describe the relationship between modeling and simulation, i.e., thinking of simulation as dynamic modeling.
2. Create a simple, formal mathematical model of a real-world situation and use that model in a simulation.
3. Differentiate among the different types of simulations, including physical simulations, human-guided simulations, and virtual reality.
4. Describe several approaches to validating models.

Learning

5. Describe Turing test and the “Chinese Room” thought experiment.
6. List the differences among the three main styles of learning: supervised, reinforcement, and unsupervised.
7. Identify examples of classification tasks, including the available input features and output to be predicted.
8. Explain the difference between inductive and deductive learning.
9. Apply the simple statistical learning algorithm such as Naive Bayesian Classifier to a classification task and measure the classifier's accuracy.
10. Differentiate between the concepts of optimal reasoning/behavior and human-like reasoning/behavior.

CSCI 320 – Algorithm Analysis

Analysis

1. Explain what is meant by “best”, “average”, and “worst” case behavior of an algorithm
2. Determine informally the time and space complexity of simple algorithms.
3. Understand the formal definition of big O
4. List and contrast standard complexity classes
5. Perform empirical studies to validate hypotheses about runtime stemming from mathematical analysis. Run algorithms on input of various sizes and compare performance
6. Apply the tools of probability to solve problems such as the average case analysis of algorithms or analyzing hashing.
7. Give examples that illustrate time-space trade-offs of algorithms.
8. Use big O notation formally to give asymptotic upper bounds on time and space complexity of algorithms.
9. Use big O notation formally to give expected case bounds on time complexity of algorithms
10. Explain the use of big omega, big theta, and little o notation to describe the amount of work done by an algorithm.
11. Use recurrence relations to determine the time complexity of recursively defined algorithms
12. Solve elementary recurrence relations, e.g., using some form of a Master Theorem
13. Describe the role of heuristics and describe the trade-offs among completeness, optimality, time complexity, and space complexity.
14. Explain the importance of locality in determining performance
15. Define the classes P and NP.
16. Explain the significance of NP-completeness.
17. Evaluate whether a heuristic for a given problem is admissible/can guarantee optimal solution.
18. Define the P-space class and its relation to the ExP class
19. Explain the significance of NP-completeness.
20. Provide examples of classic NP-complete problems.
21. Prove that a problem is NP-complete by reducing a classic known NP-complete problem to it.
22. Identify the issues impacting correctness and efficiency of a computation.

Strategies

23. For each of the above strategies (brute force, greedy, divide and conquer, backtracking, dynamic), identify a practical example to which it would apply
24. Use a divide-and-conquer algorithm to solve an appropriate problem.
25. Use dynamic programming to solve an appropriate problem.
26. Use a greedy approach to solve an appropriate problem and determine if the greedy rule chosen leads to an optimal solution
27. Implement a divide-and-conquer algorithm for solving a problem.
28. Describe various heuristic problem-solving methods
29. Describe the trade-offs between brute force and other strategies
30. Use advanced algorithmic techniques (e.g., randomization, approximation) to solve real problems.

Design

31. In the context of specific algorithms, identify the characteristics of data and/or other conditions or assumptions that lead to different behaviors
32. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data
33. Demonstrate the ability to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in a particular context
34. Apply the techniques of decomposition to break a program into smaller pieces.
35. Identify the relative strengths and weaknesses among multiple designs or implementations for a problem.
36. Choose the appropriate data structure for modeling a given problem.
37. Have facility mapping pseudocode to implementation, implementing examples of algorithmic strategies from scratch, and applying them to specific problems
38. Use a heuristic approach to solve an appropriate problem
39. Understand the mapping of real-world problems to algorithmic solutions (e.g., as graph problems, linear programs, etc.)
40. Analyze simple problem statements to identify relevant information and select appropriate processing to solve the problem.

Recursion

41. Solve a variety of basic recurrence relations.
42. Analyze a problem to determine underlying recurrence relations.
43. Use recursive backtracking to solve a problem such as navigating a maze
44. Describe the concept of recursion and give examples of its use.
45. Identify the base case and the general case of a recursively-defined problem.

Hashing, Searching, Sorting & Matching

46. Understand the implementation of hash tables, including collision avoidance and resolution
47. Implement simple search algorithms and explain the differences in their time complexities
48. Be able to implement common quadratic and O(N log N) sorting algorithms
49. Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing
50. Be able to implement a string-matching algorithm

CSCI 340 – Computer Architecture

Architecture

1. Describe the basic building blocks of computers and their role in the historical development of computer architecture.
2. Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers. This can be motivated by real-world systems, like how a car works, or libraries.
3. Recognize that hardware, VM, OS, application are additional layers of interpretation/processing.
4. Explain how the components of system architecture contribute to improving its performance.
5. Describe Amdahl's law and discuss its limitations.
6. Explain the organization of the classical von Neumann machine and its major functional units.
7. Describe the progression of computer technology components from vacuum tubes to VLSI, from mainframe computer architectures to the organization of warehouse-scale computers
8. Comprehend the trend of modern computer architectures towards multi-core and that parallelism is inherent in all hardware systems
9. Explain the implications of the "power wall" in terms of further processor performance improvements and the drive towards harnessing parallelism
10. Describe the SMP architecture and note its key features
11. Characterize the kinds of tasks that are a natural match for SIMD machines

Circuit Design

12. Design a simple logic circuit using the fundamental building blocks of logic design.
13. Use tools for capture, synthesis, and simulation to evaluate a logic design.
14. Evaluate the functional and timing diagram behavior of a simple processor implemented at the logic circuit level.
15. Articulate that there are many equivalent representations of computer functionality, including logical expressions and gates, and be able to use mathematical expressions to describe the functions of simple combinational and sequential circuits
16. Design the basic building blocks of a computer: arithmetic-logic unit (gate-level), registers (gate-level), central processing unit (register transfer-level), memory (register transfer-level)
17. Use CAD tools for capture, synthesis, and simulation to evaluate simple building blocks (e.g., arithmetic-logic unit, registers, movement between registers) of a simple computer design

Assembly Language

18. Write simple assembly language program segments.
19. Demonstrate how to map between high-level language patterns into assembly/machine language notations
20. Explain how subroutine calls are handled at the assembly level
21. Write simple programs at the assembly/machine level for string processing and manipulation

Machine Level Instructions

22. Show how fundamental high-level programming constructs are implemented at the machine-language level
23. Describe instruction level parallelism and hazards, and how they are managed in typical processor pipelines
24. Summarize how instructions are represented at both the machine level and in the context of a symbolic assembler
25. Explain different instruction formats, such as addresses per instruction and variable length vs. fixed length formats

Data Representation

26. Explain why everything is data, including instructions, in computers
27. Explain the reasons for using alternative formats to represent numerical data
28. Describe how negative integers are stored in sign-magnitude and twos-complement representations.
29. Explain how fixed-length number representations affect accuracy and precision
30. Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays.
31. Convert numerical data from one format to another.

Memory

32. Identify the main types of memory technology
33. Explain the effect of memory latency on running time
34. Describe how the use of memory hierarchy (cache, virtual memory) is used to reduce the effective memory latency
35. Compute Average Memory Access Time under a variety of memory system configurations and workload assumptions

Interrupts and I/O

36. Describe the mechanisms of how errors are detected, signaled back, and handled through the layers.

37. Explain the basic concepts of interrupts and I/O operations
38. Explain how interrupts are used to implement I/O control and data transfers
39. Identify various types of buses in a computer system
40. Describe data access from a magnetic disk drive
41. Identify interfaces needed for multimedia support, from storage, through network, to memory and display
42. Describe the advantages and limitations of RAID architectures

CSCI 440 – Operating Systems
* To be taught in the new Networking & Security class

Organization

1. Explain the objectives and functions of modern operating systems
2. Analyze the tradeoffs inherent in operating system design
3. Recognize that hardware, VM, OS, application are additional layers of interpretation/processing.
4. Explain the concept of a logical layer
5. Describe the functions of a contemporary operating system with respect to convenience, efficiency, and the ability to evolve
6. Explain the benefits of building abstract layers in hierarchical fashion
7. Defend the need for APIs and middleware
8. Describe how computing resources are used by application software and managed by system software
9. Contrast kernel and user mode in an operating system
10. Discuss networked, client-server, distributed operating systems and how they differ from single user operating systems.
11. Articulate the differences between single thread vs. multiple thread, single server vs. multiple server models, motivated by real world examples (e.g., cooking recipes, lines for multiple teller machines, couple shopping for food, wash-dry-fold, etc.).
12. Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers. This can be motivated by real-world systems, like how a car works, or libraries.
13. Describe how an instruction is executed in a classical von Neumann machine, with extensions for threads, multiprocessor synchronization, and SIMD execution

Concurrency

14. Demonstrate using a high-level programming language how to prevent a race condition from occurring and how to handle an exception.
15. Use mutual exclusion to avoid a given race condition
16. Distinguish multiple sufficient programming constructs for synchronization that may be inter-implementable but have complementary advantages
17. Distinguish data races from higher level races
18. Describe the need for concurrency within the framework of an operating system
19. Demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks
20. Summarize the range of mechanisms that can be employed at the operating system level to realize concurrent systems and describe the benefits of each
21. Explain the different states that a task may pass through and the data structures needed to support the management of many tasks
22. Summarize techniques for achieving synchronization in an operating system (e.g., describe how to implement a semaphore using OS primitives)
23. Describe reasons for using interrupts, dispatching, and context switching to support concurrency in an operating system
24. Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems, such as priority, performance comparison, and fair-share schemes.
25. Describe relationships between scheduling algorithms and application domains
26. Discuss the types of processor scheduling such as short-term, medium-term, long-term, and I/O
27. Describe the difference between processes and threads.
28. Compare and contrast static and dynamic approaches to real-time scheduling
29. Discuss the need for preemption and deadline scheduling
30. Identify ways that the logic embodied in scheduling algorithms are applicable to other domains, such as disk I/O, network scheduling, project scheduling, and problems beyond computing
31. Give an example of an ordering of accesses among concurrent activities that is not sequentially consistent
32. Give an example of a scenario in which blocking message sends can deadlock
33. Explain when and why multicast or event-based messaging can be preferable to alternatives
34. Write a program that correctly terminates when all of a set of concurrent tasks have completed
35. Use a properly synchronized queue to buffer data passed among activities
36. Explain why checks for preconditions, and actions based on these checks, must share the same unit of atomicity to be effective
37. Write a test program that can reveal a concurrent programming error; for example, missing an update when two activities both try to increment a variable
38. Describe at least one design technique for avoiding liveness failures in programs using multiple locks or

39. Describe at least one design technique for avoiding liveness failures in programs using multiple locks or semaphores
40. Describe the relative merits of optimistic versus conservative concurrency control under different rates of contention among updates
41. Give an example of a scenario in which an attempted optimistic update may never complete
42. Define “critical path”, “work”, and “span”
43. Compute the work and span, and determine the critical path with respect to a parallel execution diagram
44. Define “speed-up” and explain the notion of an algorithm’s scalability in this regard
45. Define how finite computer resources (e.g., processor share, memory, storage and network bandwidth) are managed by their careful allocation to existing entities.
46. Describe the scheduling algorithms by which resources are allocated to competing entities, and the figures of merit by which these algorithms are evaluated, such as fairness.
47. Implement simple schedule algorithms.
48. Measure figures of merit of alternative scheduler implementations.
49. Explain why it is important to isolate and protect the execution of individual programs and environments that share common underlying resources, including the processor, memory, storage, and network access.
50. Use semaphores or condition variables to block threads until a necessary precondition holds
51. Rewrite a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions

Memory & I/O

52. Explain the differences between shared and distributed memory
53. Distinguish using computational resources for a faster answer from managing efficient access to a shared resource
54. Explain the use of a device list and driver I/O queue
55. Identify the main types of memory technology
56. Describe the principles of memory management
57. Explain memory hierarchy and cost-performance trade-offs.
58. Explain the effect of memory latency on running time
59. Explain the workings of a system with virtual memory management
60. Describe how the use of memory hierarchy (cache, virtual memory) is used to reduce the effective memory latency
61. Summarize the principles of virtual memory as applied to caching and paging.
62. Evaluate the trade-offs in terms of memory size (main memory, cache memory, auxiliary memory) and processor speed
63. Defend the different ways of allocating memory to tasks, citing the relative merits of each
64. Describe the reason for and use of cache memory (performance and proximity, different dimension of how caches complicate isolation and VM abstraction)
65. Discuss the concept of thrashing, both in terms of the reasons it occurs and the techniques used to recognize and manage the problem
66. Describe why things that are close in space take less time to access
67. Calculate average memory access time and describe the tradeoffs in memory hierarchy performance in terms of capacity, miss/hit rate, and access time
68. Explain how programming language implementations typically organize memory into global data, text, heap, and stack sections and how features such as recursion and memory management map to this memory model.
69. Reason about memory leaks, dangling-pointer dereferences, and the benefits and limitations of garbage collection.
70. Explain how programming language implementations typically organize memory into global data, text, heap, and stack sections and how features such as recursion and memory management map to this memory model.
71. Describe data access from a magnetic disk drive
72. Describe the advantages and disadvantages of direct memory access and discuss the circumstances in which its use is warranted
73. Explain the concept of virtual memory and how it is realized in hardware and software

Interrupts

74. Describe the mechanisms of how errors are detected, signaled back, and handled through the layers.
75. Construct a simple program using methods of layering, error detection and recovery, and reflection of error status across layers.
76. Discuss the advantages and disadvantages of using interrupt processing.
77. Explain how interrupts are used to implement I/O control and data transfers
78. Explain the basic concepts of interrupts and I/O operations

79. Explain buffering and describe strategies for implementing it

Security

80. Identify potential threats to operating systems and the security features design to guard against them.
81. Defend the need for protection and security in an OS (cross reference IAS/Security Architecture and Systems Administration/Investigating Operating Systems Security for various systems)
82. Summarize the features and limitations of an operating system used to provide protection and security
83. Explain the mechanisms available in an OS to control access to resources
84. Carry out simple system administration tasks according to a security policy, for example creating accounts, setting permissions, applying patches, and arranging for regular backups *
85. Explain the concept of identity management and its importance

Parallel Processing

86. Define the differences between the concepts of Instruction Parallelism, Data Parallelism, Thread Parallelism/Multitasking, Task/Request Parallelism.*
87. Write a simple sequential program and a simple parallel version of the same program. *
88. Evaluate performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved.*
89. For a given program, distinguish between its sequential and parallel execution, and the performance implications thereof.*
90. Demonstrate on an execution time line that parallelism events and operations can take place simultaneously (i.e., at the same time). Explain how work can be performed in less elapsed time if this can be exploited.*
91. Explain other uses of parallelism, such as for reliability/redundancy of execution.*
92. Write more than one parallel program (e.g., one simple parallel program in more than one parallel programming paradigm; a simple parallel program that manages shared resources through synchronization primitives; a simple parallel program that performs simultaneous operation on partitioned data through task parallel (e.g., parallel search terms; a simple parallel program that performs step-by-step pipeline processing through message passing).*
93. Use performance tools to measure speed-up achieved by parallel programs in terms of both problem size and number of resources.*
94. Explain why synchronization is necessary in a specific parallel program*
95. Identify independent tasks in a program that may be parallelized*
96. Characterize features of a workload that allow or prevent it from being naturally parallelized*
97. Implement a parallel divide-and-conquer and/or graph algorithm and empirically measure its performance relative to its sequential analog *
98. Write a correct and scalable parallel algorithm *
99. Parallelize an algorithm by applying task-based decomposition *
100. Parallelize an algorithm by applying data-parallel decomposition *
101. Decompose a problem (e.g., counting the number of occurrences of some word in a document) via map and reduce operations*

System Programming

102. Find bugs in a layered program by using tools for program tracing, single stepping, and debugging.
103. Design and conduct a performance-oriented experiment, e.g., benchmark a parallel program with different data sets in order to iteratively improve its performance. *
104. Use software tools to profile and measure program performance.

Virtualization

105. Describe how the concept of indirection can create the illusion of a dedicated machine and its resources even when physically shared among multiple programs and environments.
106. Measure the performance of two application instances running on separate virtual machines, and determine the effect of performance isolation.
107. Identify the relationship between the physical hardware and the virtual devices maintained by the operating system
108. Discuss hypervisors and the need for them in conjunction with different types of hypervisors

Client-Server Model

109. Implement a simple client-server socket-based application. *
110. Describe the differences between synchronous and asynchronous communication*
111. Describe the three levels of software in the client-server model*

Protocols

112. Design and implement a simple reliable protocol

CIS 250 – Introduction to Information Systems

Human-Computer Interaction (HCI)

1. Discuss why human-centered software development is important
2. Develop and use a conceptual vocabulary for analyzing human interaction with software: affordance, conceptual model, feedback, and so forth
3. Define a user-centered design process that explicitly recognizes that the user is not like the developer or her acquaintances
4. Identify developers' assumptions and values embedded in hardware and software design, especially as they pertain to usability for diverse populations including under-represented populations and the disabled.
5. Students should be able to apply the principles of HCI foundations to: Conduct a quantitative evaluation and discuss/report the results.
6. Students should be able to apply the principles of HCI foundations to: Discuss at least one national or international user interface design standard
7. Identify the social implications of ergonomic devices and the workplace environment to people's health.

Societal Issues

8. Describe how software can interact with and participate in various systems including information management, embedded, process control, and communications systems.
9. Summarize the basic precepts of psychological and social interaction
10. Describe positive and negative ways in which computer technology (networks, mobile computing, cloud computing) alters modes of social interaction at the personal level.
11. Interpret the social context of a given design and its implementation.
12. Evaluate the efficacy of a given design and implementation using empirical data.
13. Investigate the implications of social media on individualism versus collectivism and culture.
14. Identify ways to be a sustainable practitioner
15. Illustrate global social and environmental impacts of computer use and disposal (e-waste)
16. Analyze a global computing issue, observing the role of professionals and government officials in managing this problem.
17. Describe ways in which professionals may contribute to public policy.
18. Discuss how Internet access serves as a liberating force for people living under oppressive forms of government; explain how limits on Internet access are used as tools of political and social repression.
19. Analyze the pros and cons of reliance on computing in the implementation of democracy (e.g. delivery of social services, electronic voting).
20. Describe the impact of the under-representation of diverse populations in the computing profession (e.g., industry culture, product diversity).
21. Describe the environmental impacts of design choices within the field of computing that relate to algorithm design, operating system design, networking design, database design, etc.
22. Investigate the social and environmental impacts of new system designs through projects.

Ethics & Professional Conduct

23. Illustrate the use of example and analogy in ethical argument.
24. Evaluate ethical/social tradeoffs in technical decisions.
25. Recognize the ethical responsibility of ensuring software correctness, reliability and safety.
26. Describe the strengths and weaknesses of relevant professional codes as expressions of professionalism and guides to decision-making.
27. Identify progressive stages in a whistle-blowing incident.
28. Investigate forms of harassment and discrimination and avenues of assistance
29. Describe the consequences of inappropriate professional behavior.
30. Develop a computer usage/acceptable use policy with enforcement measures.

Intellectual Property

31. Discuss the philosophical bases of intellectual property.
32. Discuss the rationale for the legal protection of intellectual property.
33. Interpret the intent and implementation of software licensing.
34. Discuss the issues involved in securing software patents.
35. Characterize and contrast the concepts of copyright, patenting and trademarks.

Privacy

36. Discuss the philosophical basis for the legal protection of personal privacy.
37. Evaluate solutions to privacy threats in transactional databases and data warehouses.
38. Recognize the fundamental role of data collection in the implementation of pervasive surveillance

systems (e.g., RFID, face recognition, toll collection, mobile computing).

39. Recognize the ramifications of differential privacy.
40. Investigate the impact of technological solutions to privacy problems.
41. Investigate the implications of context awareness in ubiquitous computing systems.

Professional Development

42. Describe the mechanisms that typically exist for a professional to keep up-to-date.
43. Examine various forms of professional credentialing

CIS 450 – Software Engineering

Communication & Teamwork

1. Write clear, concise, and accurate technical documents following well-defined standards for format and for including appropriate tables, figures, and references.
2. Evaluate written technical documentation to detect problems of various kinds.
3. Develop and deliver a good quality formal presentation.
4. Plan interactions (e.g. virtual, face-to-face, shared documents) with others in which they are able to get their point across, and are also able to listen carefully and appreciate the points of others, even when they disagree, and are able to convey to others that they have heard.
5. Describe the strengths and weaknesses of various forms of communication (e.g. virtual, face-to-face, shared documents)
6. Examine appropriate measures used to communicate with stakeholders involved in a project.
7. Compare and contrast various collaboration tools.
8. Identify ethical issues that arise in software development and determine how to address them technically and ethically.
9. Identify behaviors that contribute to the effective functioning of a team.
10. Create and follow an agenda for a team meeting.
11. Identify and justify necessary roles in a software development team.
12. Understand the sources, hazards, and potential benefits of team conflict.
13. Apply a conflict resolution strategy in a team setting.

Development Tools

14. Describe the issues that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing.
15. Describe the difference between centralized and distributed software configuration management.

Software Life Cycle

16. Describe the difference between principles of the waterfall model and models using iterations.
17. Describe the different practices that are key components of various process models.
18. Differentiate among the phases of software development.
19. Describe how programming in the large differs from individual efforts with respect to understanding a large code base, code reading, understanding builds, and understanding context of changes.
20. Identify the principal issues associated with software evolution and explain their impact on the software life cycle.
21. Estimate the impact of a change request to an existing product of medium size.
22. Explain the concept of a software life cycle and provide an example, illustrating its phases including the deliverables that are produced.
23. Compare several common process models with respect to their value for development of particular classes of software systems taking into account issues such as requirement stability, size, and non-functional characteristics.
24. List several examples of software risks.
25. Describe the impact of risk in a software development life cycle.
26. Describe different categories of risk in software systems.
27. Explain the problems that exist in achieving very high levels of reliability.
28. Describe how software reliability contributes to system reliability
29. List approaches to minimizing faults that can be applied at each stage of the software lifecycle.
30. Summarize the phases of software development and compare several common lifecycle models.
31. Describe software development best practices for minimizing vulnerabilities in programming code.

Planning

32. Interpret a given requirements model for a simple software system.
33. Conduct a review of a set of software requirements to determine the quality of the requirements with respect to the characteristics of good requirements.
34. Evaluate stakeholder positions in a given situation.
35. Use an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required.
36. Describe the fundamental challenges of and common techniques used for requirements elicitation.
37. Identify both functional and non-functional requirements in a given requirements specification for a software system.

Testing

38. Create and conduct a simple usability test for an existing software application
39. Create a unit test plan for a medium-size code segment.

40. Outline the process of regression testing and its role in release management.
41. Distinguish between program validation and verification.
42. Describe the role that tools can play in the validation of software.
43. Describe and distinguish among the different types and levels of testing (unit, integration, systems, and acceptance).
44. Describe techniques for identifying significant test cases for unit, integration, and system testing.

Maintenance

45. Understand the need for the ability to update software to fix security vulnerabilities
46. Discuss the challenges of evolving systems in a changing environment.

Intellectual Property

47. Discuss the philosophical bases of intellectual property.
48. Discuss the rationale for the legal protection of intellectual property.
49. Describe legislation aimed at digital copyright infringements.
50. Critique legislation aimed at digital copyright infringements
51. Identify contemporary examples of intangible digital intellectual property
52. Justify uses of copyrighted materials.
53. Evaluate the ethical issues inherent in various plagiarism detection mechanisms.
54. Interpret the intent and implementation of software licensing.
55. Discuss the issues involved in securing software patents.
56. Characterize and contrast the concepts of copyright, patenting and trademarks.

CSCI 140 – Introduction to Web API

1. Understand the risks with misusing interfaces with third-party code and how to correctly use third-party code
2. Design and Implement a simple web application
3. Describe the constraints that the web puts on developers
4. Compare and contrast web programming with general purpose programming
5. Discuss how web standards impact software development
6. Review an existing web application against a current web standard

CSCI 435 – Compiler Design

1. Distinguish a language definition (what constructs mean) from a particular language implementation (compiler vs. interpreter, run-time representation of data objects, etc.).
2. Process some representation of code for some purpose, such as an interpreter, an expression optimizer, a documentation generator, etc.
3. Use declarative tools to generate parsers and scanners.
4. Identify key issues in syntax definitions: ambiguity, associativity, and precedence.

CIS 325 – Data Communications (CSCI 325)

Network Architecture

1. Articulate the organization of the Internet
2. List and define the appropriate network terminology
3. Describe the layered structure of a typical networked architecture.
4. Identify the different levels of complexity in a network (edges, core, etc.)
5. List the differences and the relations between names and addresses in a network
6. Define the principles behind naming schemes and resource location
7. Articulate the concept of strong vs. weak scaling, i.e., how performance is affected by scale of problem vs. scale of resources to solve the problem. This can be motivated by the simple, real-world examples.
8. Describe the issues and approaches to testing distributed and parallel systems.

Ethernet Networks

9. Compare common network organizations, such as ethernet/bus, ring, switched vs. routed.
10. Describe how frames are forwarded in an Ethernet network
11. Identify the differences between IP and Ethernet.
12. Describe the steps used in one common approach to the multiple access problem
13. Describe the interrelations between IP and Ethernet

Wireless Networks

14. Describe the organization of a wireless network
15. Describe how wireless networks support mobile users

Protocols

16. Describe the operation of reliable delivery protocols.
17. List the factors that affect the performance of reliable delivery protocols

Network Traffic

18. Describe how resources can be allocated in a network
19. Describe the congestion problem in a large network
20. Compare and contrast the fixed and dynamic allocation techniques
21. Compare and contrast current approaches to congestion
22. Describe the organization of the network layer
23. Describe how packets are forwarded in an IP networks
24. List the scalability benefits of hierarchical addressing

Client-Server

25. Implement a simple client-server socket-based application.
26. Describe the three levels of software in the client-server model

Security

27. Understand the tradeoffs and balancing of key security properties (Confidentiality, Integrity, Availability)
28. Understand the concepts of risk, threats, vulnerabilities and attack vectors (including the fact that there is no such thing as perfect security)
29. Understand the concept of authentication, authorization, access control
30. Understand the concept of trust and trustworthiness
31. Be able to recognize that there are important ethical issues to consider in computer security, including ethical issues associated with fixing or not fixing vulnerabilities and disclosing or not disclosing vulnerabilities
32. Describe the principle of least privilege and isolation and apply to system design
33. Understand the principle of fail-safe and deny-by-default
34. Understand not to rely on the secrecy of design for security (but also that open design alone does not imply security)
35. Understand the goals of end-to-end data security
36. Understand the benefits of having multiple layers of defenses
37. Understand that security has to be a consideration from the point of initial design and throughout the lifecycle of a product
38. Understanding that security imposes costs and tradeoffs
39. Describe the different categories of network threats and attacks.
40. Describe the architecture for public and private key cryptography and how PKI supports network security.
41. Describe virtues and limitations of security technologies at each layer of the network stack.
42. Identify the appropriate defense mechanism(s) and its limitations given a network threat
43. Understand security properties and limitations of other non-wired networks

44. Describe the concept of mediation and the principle of complete mediation
45. Know to use standard components for security operations, instead of re-inventing fundamentals operations
46. Understand the concept of trusted computing including trusted computing base and attack surface and the principle of minimizing trusted computing base
47. Understand the importance of usability in security mechanism design
48. Understand that security does not compose by default; security issues can arise at boundaries between multiple components
49. Understand the different roles of prevention mechanisms and detection/deterrence mechanisms
50. Describe likely attacker types against a particular system
51. Understand malware species and the virus and limitations of malware countermeasures (e.g., signature-based detection, behavioral detection)
52. Identify instances of social engineering attacks and Denial of Service attacks
53. Understand the concepts of side channels and covert channels and their differences
54. Discuss the manner in which Denial of Service attacks can be identified and mitigated.
55. Describe risks to privacy and anonymity in commonly used applications
56. Describe the basic properties of bandwidth, latency, scalability and granularity.
57. Explain the concept of identity management and its importance
58. Explain the concepts of phishing and spear phishing, and how to recognize them
59. Design a user interface for a security mechanism
60. Configure a network for security

CIS 350 – Database Management

Information Management

1. Describe how humans gain access to information and data to support their needs
2. Understand advantages and disadvantages of central organizational control over data
3. Identify the careers/roles associated with information management (e.g., database administrator, data modeler, application developer, end-user).
4. Compare and contrast information with data and knowledge
5. Demonstrate uses of explicitly stored metadata/schema associated with data
6. Identify issues of data persistence for an organization
7. Explain the characteristics that distinguish the database approach from the traditional approach of programming with data files
8. Critique/defend a small- to medium-size information application with regard to its satisfying real user information needs
9. Describe several technical solutions to the problems related to information privacy, integrity, security, and preservation
10. Understand the most common designs for core database system components including the query optimizer, query executor, storage manager, access methods, and transaction processor.
11. Cite the basic goals, functions, models, components, applications, and social impact of database systems
12. Describe the components of a database system and give examples of their use
13. Identify major DBMS functions and describe their role in a database system
14. Explain the concept of data independence and its importance in a database system
15. Describe how various types of content cover the notions of structure and/or of stream (sequence), e.g., documents, multimedia, tables
16. Explain measures of efficiency (throughput, response time) and effectiveness (recall, precision)
17. Identify approaches that scale up to globally networked systems
18. Identify vulnerabilities and failure scenarios in common forms of information systems
19. Identify all of the data, information, and knowledge elements and related organizations, for a computational science application.
20. Describe how to represent data and information for processing.
21. Describe typical user requirements regarding that data, information, and knowledge.
22. Select a suitable system or software implementation to manage data, information, and knowledge.
23. Use standard APIs and tools to create visual displays of data, including graphs, charts, tables, and histograms.
24. Describe several approaches to using a computer as a means for interacting with and processing data.
25. Extract useful information from a dataset.
26. Analyze and select visualization techniques for specific problems.
27. Describe issues related to scaling data analysis from small to large data sets.
28. Describe major approaches to storing and processing large volumes of data
29. Explain basic information storage and retrieval concepts
30. Explain the concepts of records, record types, and files, as well as the different techniques for placing file records on disk

Modeling

31. Categorize data models based on the types of concepts that they provide to describe the database structure and their usage, for example, use of conceptual, spreadsheet, physical, and representational data models
32. Describe the modeling concepts and notation of widely used modeling notation (e.g., ERD notation, and UML), including their use in data modeling.
33. Define the fundamental terminology used in the relational data model
34. Describe the basic principles of the relational data model
35. Apply the modeling concepts and notation of the relational data model.
36. Describe the differences between relational and semi-structured data models
37. Give a semi-structured equivalent (e.g., in DTD or XML Schema) for a given relational schema
38. Prepare a relational schema from a conceptual model developed using the entity- relationship model
39. Explain and demonstrate the concepts of entity integrity constraint and referential integrity constraint (including definition of the concept of a foreign key)
40. Determine the functional dependency between two or more attributes that are a subset of a relation
41. Connect constraints expressed as primary key and foreign key, with functional dependencies
42. Compute the closure of a set of attributes under given functional dependencies

43. Determine whether or not a set of attributes form a superkey and/or candidate key for a relation with given functional dependencies
44. Evaluate a proposed decomposition, to say whether or not it has lossless-join and dependency-preservation
45. Describe what is meant by BCNF, PJNF, 5NF
46. Explain the impact of normalization on the efficiency of database operations especially query optimization
47. Describe what is a multi-valued dependency and what type of constraints it specifies

SQL

48. Use a declarative query language to elicit information from a database.
49. Explain uses of declarative queries
50. Give a declarative version for a navigational query
51. Create a relational database schema in SQL that incorporates key, entity integrity, and referential integrity constraints
52. Demonstrate data definition in SQL and retrieving information from a database using the SQL SELECT statement
53. Demonstrate use of the relational algebra operations from mathematical set theory (union, intersection, difference, and Cartesian product) and the relational algebra operations developed specifically for relational databases (select (restrict), project, join, and division)
54. Demonstrate queries in the relational algebra
55. Demonstrate queries in the tuple relational calculus

Transaction Management

56. Create a transaction by embedding SQL into an application program
57. Explain the concept of implicit commits
58. Describe the issues specific to efficient transaction execution
59. Explain when and why rollback is needed and how logging assures proper rollback
60. Explain the effect of different isolation levels on the concurrency control mechanisms
61. Choose the proper isolation level for implementing a specified transaction protocol
62. Identify appropriate transaction boundaries in application programs

Software Patterns and Design

Software Tools

1. Use a defect tracking tool to manage software defects in a small software project.
2. Identify configuration items and use a source code control tool in a small team-based project.
3. Demonstrate the capability to use software tools in support of the development of a software product of medium size.

Design

4. List the key components of a use case or similar description of some behavior that is required for a system and discuss their role in the requirements engineering process.
5. Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design. [Application]
6. Construct models of the design of a simple software system that are appropriate for the paradigm used to design it.
7. For the design of a simple software system within the context of a single design paradigm, describe the software architecture of that system.
8. Create appropriate models for the structure and behavior of software products from their requirements specifications.

Implementation

9. Students should be able to apply the principles of HCI foundations to: Create a simple application, together with help & documentation, that supports a user interface
10. Select and use a defined coding standard in a small software project.
11. Compare and contrast integration strategies including top-down, bottom-up, and sandwich integration.
12. Describe the process of analyzing and implementing changes to code base developed for a specific project.
13. Describe the process of analyzing and implementing changes to a large existing code base.

Testing

14. Create a unit test plan for a medium-size code segment.
15. Undertake, as part of a team activity, an inspection of a medium-size code segment.

Software Patterns

16. List commonly encountered patterns of how computations are organized.
17. Within the context of a single design paradigm, describe one or more design patterns that could be applicable to the design of a simple software system.
18. Apply simple examples of patterns in a software design.

Software Design

19. Interpret a given requirements model for a simple software system.
20. Articulate design principles including separation of concerns, information hiding, coupling and cohesion, and encapsulation.
21. Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design.
22. Construct models of the design of a simple software system that are appropriate for the paradigm used to design it.
23. For the design of a simple software system within the context of a single design paradigm, describe the software architecture of that system.
24. List the key components of a class diagram or similar description of the data that a system is required to handle.
25. Identify weaknesses in a given simple design, and removed them through refactoring.
26. For a simple system suitable for a given scenario, discuss and select an appropriate design paradigm.
27. Create appropriate models for the structure and behavior of software products from their requirements specifications.
28. Explain the relationships between the requirements for a software product and the designed structure and behavior, in terms of the appropriate models and transformations of them.
29. Given a high-level design, identify the software architecture by differentiating among common software architectures such as 3-tier, pipe-and-filter, and client-server.
30. Investigate the impact of software architectures selection on the design of a simple system.
31. Select suitable components for use in the design of a software product.
32. Explain how suitable components might need to be adapted for use in the design of a software product.
33. Design a contract for a typical small software component for use in a given system.
34. State and apply the principles of least privilege and fail-safe defaults.

Computer Graphics

1. Describe the differences between lossy and lossless image compression techniques, for example as reflected in common graphics image file formats such as JPG, PNG, and GIF.
2. Identify common uses of computer graphics.
3. Explain in general terms how analog signals can be reasonably represented by discrete samples, for example, how images can be represented by pixels.
4. Construct a simple user interface using a standard graphics API.
5. Describe color models and their use in graphics display devices.
6. Describe the tradeoffs between storing information vs storing enough information to reproduce the information, as in the difference between vector and raster rendering.
7. Describe the basic process of producing continuous motion from a sequence of discrete frames (sometimes called “flicker fusion”).
8. Describe how double-buffering can remove flicker from animation.
9. Write event handlers for use in reactive systems, such as GUIs.

Cryptography

1. Describe the purpose of Cryptography and list ways it is used in data communications.
2. Define the following terms: Cipher, Cryptanalysis, Cryptographic Algorithm, and Cryptology and describe the two basic methods (ciphers) for transforming plain text in cipher text.
3. Discuss the importance of prime numbers in cryptography and explain their use in cryptographic algorithms.
4. Understand how to measure entropy and how to generate cryptographic randomness.
5. Demonstrate how Public Key Infrastructure supports digital signing and encryption and discuss the limitations/vulnerabilities.
6. Understand the role of random numbers in security, beyond just cryptography (e.g., password generation, randomized algorithms to avoid algorithmic denial of service attacks)
7. Understand the cryptographic primitives and their basic properties.
8. Understand public-key primitives and their applications.
9. Understand how key exchange protocols work and how they fail.
10. Understand cryptographic protocols and their properties.
11. Describe real-world applications of cryptographic primitives and protocols.
12. Understand precise security definitions, attacker capabilities and goals.