Introductory Sequence Review Committee Report w/ Addendum

The 2006 Faculty Retreat Committee on Undergraduate Education was formed to examine various problems, such as our declining enrollments, and to make recommendations to improve the situation. Based on their report and subsequent discussion at the department retreat, a new committee, headed by Dr. Porter with support from Drs. Bederson, Jacob (ECE), Reggia, Samet, Sussman, and Ms. Plane, Mr. Emad and Mr. Padua-Perez, was formed to evaluate our pre-400-level courses and to initiate discussion on possible improvements. This committee issued an initial report which appears below in Sections 1–4.

large UPDATE: 3/08. Based in part on this initial report, several changes were and are being made to several courses. A subset of the original committee – Porter, Hicks, Plane, Reggia and Samet, with additional help from Michelle Hugue and J.P. Dickerson – have examined some of these changes. We are therefore supplementing our initial report with new material (Sections 5 and 6) describing and analyzing these changes.

1 Specific Charges to the Committee

Dr. Davis’ specific charges to the committee included:

Evaluate the success of the new introductory sequence, and suggest any modifications that should be made to that sequence as well as to our 300 level courses (since 330 and 311 have also been re-designed).

Look into some of the issues raised at the retreat, such possible overlap between 212 and 311, possible problems with the intellectual content of 212, disconnects between faculty assumptions about what students will know from the 100-300 sequence and what they actually know when they show up in a 400 level course.

Take a second look at what we are doing - work with the field committees and ECE to get some well defined expectations of what knowledge and skills students coming out of the 300 level courses should have, map those to the material we present in those courses, and suggest changes to the curricula for broader faculty discussion.

Solicit input from ECE as our courses form the core of their program.

2 Executive Summary

In the last few years our department has taken many steps to improve the undergraduate curriculum. Many faculty members working alone or in small groups have expended considerable effort and energy in this process. Overall, these steps substantially improved the quality of our curriculum and we commend all those who helped in this effort.

As often happens, however, these changes are both imperfect – they created new problems – and incomplete – they haven’t addressed all the old problems. For example, some quite capable students have complained that our introductory courses are intellectually overwhelming (new problem). Some professors on the other hand complain that students arriving at the 400-level are inadequately prepared (a very old complaint). We note that we were not able to “measure” the success of our recent changes, because no formal mechanisms are in place to support this.

Nevertheless, in trying to better understand these issues the committee finds that while individual courses teach important computer science material, there are numerous problems with how these courses support the overall curriculum and the educational development of our students. Specific problems include excessive overlap between some courses, lack of information flow between some courses, and an inappropriate quality and complexity of concepts in some early courses.

Therefore, the committee suggests that work continue on defining the global goals and outcomes we want our curriculum to achieve, that efforts continue to fine tune our course offerings to better support these goals and outcomes, and that mechanisms be put into place to measure the success of recent and future changes.

3 Methodology

To carry out our tasks the committee surveyed a variety of instructors who had recently taught either introductory courses or had taught one of our more popular 400-level courses. From these surveys we collected information as to the knowledge, concepts and experiences instructors expect students to have upon entering these courses.
We also collected information about the knowledge, concepts and experiences instructors expected to have after successfully completing these courses.

We then assembled this information, cross-referencing it with other information sources such as recent course web pages, discussions with other instructors, and the official prerequisite structure. We did not, however, check the information for completeness, and as mentioned below, some information does appear to have been left out. From here we built an intellectual road map tracking the flow of knowledge, concepts and experiences through our curriculum. This effort started with the 400-level courses and worked backward matching knowledge, skills and experiences expected in one course with their outcomes in previous courses.

The map and its raw inputs are available at www.cs.umd.edu/~aporter/Committee/coursemap.html. Theses materials include the raw survey text and a dot-formatted graphical layout.

Although preliminary, our analysis of the resulting intellectual road map yields several interesting insights, which we offer as a starting point for future investigation.

4 Preliminary Findings

This section presents initial findings based on our analysis of the course road map. In the following discussion the term, “IN” refers to the set of concepts, skills and experiences expected upon entry to a course; the term “OUT” refers to the set of concepts, skills and experiences assumed to have been learned upon successful completion of a course.

4.1 Structural Analysis

This section presents some findings based only on the structure of the map (i.e., without a deeper analysis of the actual information flow). Note that these findings appear in no particular order.

1. Some IN requirements have no sources. We found a small number, six or so, of IN requirements that have no identified source. These mostly appear to be oversights in writing out requirements. The one exception might be CMSC 417 which expects some OS knowledge that isn’t guaranteed by the structure.

   Our initial observation is that our 100- to 300-level courses are covering (although in some cases superficially) the concepts needed by the 400-level.

2. Some OUT concepts have no sinks. We found that some OUT concepts do not feed any IN requirements directly. We found this mostly in CMSC 131, CMSC 132, CMSC 212 and CMSC 330. In general, this set is composed of (a) abstract concepts such as 330’s “broader view of language design choices”, (b) specific technologies that are not widely used in later courses, such as 132’s “describing object-oriented designs using class diagrams”, and initial exposure to topics that are covered more deeply later, such as 131’s “Algorithms & complexity (big-O)”.

   One initial observation is that these orphan concepts should be looked at more carefully to better understand their value to the overall curriculum. We note that everything we are teaching is legitimate computer science; everything we are teaching is supported by one or more passionate champions. Nevertheless, undergraduate programs are time constrained and orphan concepts are not on the critical path. This clearly does not mean that such concepts are unimportant; it does not mean that they shouldn’t be taught. It does, however, suggest that should some content need to be cut in support of the overall program, then these concepts might be looked at first.

3. CMSC 212 feeds a limited number of concepts to a limited number of courses. The remaining OUT concepts have no sinks. CMSC 212 is mostly used by courses in which need students to write C programs using pointers and actively managing memory. Some other courses use it a general introduction to systems programming issues. In particular, the course projects have a very low-level systems orientation.

   Our initial observation is that this course has a core of information used by some later courses, while the rest is barely used (which – again, does not mean it’s unimportant). In addition, the course serves different constituencies. That is, it delivers C programming to one group of courses, while it delivers systems knowledge to another. It’s not clear that all students need all of the content delivered by this course, especially if they are not taking CMSC 412 or CMSC 417. Thus, it’s not clear that this course is sufficiently focused.
4. CMSC 330 is a prerequisite to several courses, but provides little or nothing directly to them. This applies to 411, 412, 414, 417, 420, 421, 430, 434 and 435. One possible exception to this is CMSC 433.

Our initial observation, supported by discussion with various faculty, is that the prerequisite connection between 330 and other courses currently serves only as a speed bump, giving the students time to gain some maturity. This may represent a wasted opportunity to provide more direct support to other classes. Some of this situation may stem from a shift in the course’s focus over the years. For example, 15 years ago, the course spent a significant amount of time on low-level language implementation which may have better supported later programming courses with low-level programming needs. Today’s course spends more time on higher level programming language issues.

4.2 Content Analysis

This section present some of our findings based an analysis of the map and the specific information flowing between courses. Again the presentation order has no significance.

1. Our student’s average time to graduate is now approaching five years. Some contributors to that include the length of the intro course sequence and our serial prerequisite structure. The 3-semester intro sequence is long and some of initial content might be too easy for well-prepared students. Also, the various math prerequisites for CMSC 131, CMSC 132 and CMSC 212 might not be strictly needed before taking these classes. Finally, it is not clear that each 300-level class needs both CMSC 212 and CMSC 250 as prerequisites.

Our initial observations are that we might want to consider more incentives for some students to place out of CMSC 131. We should consider whether CMSC 131 could be taken concurrently with MATH115, CMSC 132 concurrently with MATH140, and CMSC212 concurrently with MATH141. Finally, we should look at whether all 300 level courses need both 212 and 250 as prerequisites. Possibly we could unserialze some of these. For example, it might be feasible to take CMSC 311 and CMSC 250 concurrently.

2. Looking at the course content it appears that there are 3 logical content threads through the curriculum. They are Theory: (250 → 351), Systems (212 → 311) and Software (212 → 330).

Our initial observation is that, as mentioned earlier, CMSC 212 serves 2 different roles, which dilutes its focus and damages its intellectual coherency. We should consider either separating the low-level C programming content from the OS/System content or lessening the emphasis on the OS/System content.

3. There is considerable overlap in the content of CMSC 212 and CMSC 311 and the content of CMSC 311 is inconsistent from semester to semester.

Our initial observations are that in light of the previous point, it seems that CMSC 212 and CMSC 311 need to reviewed in terms of their content and relationships with other courses.

Currently CMSC 212 pushes into issues of machine architecture and assembly language programming. This same material gets covered in CMSC 311.

Recent offerings of CMSC311 have differed greatly in their fidelity to the course description, which focuses the course on “Computer Organization: Introduction to assembly language. Design of digital logic circuits. Organization of central processors, including instruction sets, register transfer operations, control, microprogramming data representation, and arithmetic algorithms. Memory and input/output organization”.

Our initial observation is that CMSC 311 needs to be revisited to determine what we want to teach in that course and to determine how it integrates with CMSC 212. We need ensure that all instructors are aware of the course’s purpose and relationship to other courses, such as CMSC 212 and CMSC 411.

4.3 Pedagogical Considerations

This section details some impressions drawn from the previous analyses, but not necessarily supported by concrete data found in the road map.
1. Abstraction vs. Implementation. CMSC 131 and CMSC 132 introduce a wide variety of fairly high-level programming concepts. Immediately after this, CMSC 212 introduces very low level programming and OS/Systems concepts.

Our initial observation is that the mental shift from CMSC 131 and CMSC 132 to CMSC 212 is too far-reaching and too abrupt. That is, our current approach goes from teaching high level computing concepts, such as string processing to teaching low level systems programming concepts, like writing an assembler, effectively overnight. We should consider ways to smooth this transition, either through better integration of abstraction and implementation (e.g., through small programming assignments, quizzes, etc., aimed at highlighting these connections), or by redistributing content in these courses.

2. Developmentally appropriate content. CMSC 131 and CMSC 132 cram a breathtakingly wide variety of advanced material into the first 2 semesters of our program. For example, CMSC 131 touches on “exception handling, generics, algorithmic complexity, multiple inheritance, sorting, deep vs. shallow array copying, etc.” CMSC 132 touches on “priority queues, Huffman trees, graph algorithms, Java serializability, etc.”

We question whether at least some of this material is too advanced for the 1st and 2nd introductory courses. Alternatively, given that this information can’t be taught at any depth, we wonder whether this approach makes the best use of teaching time. While it certainly gives students a quick overview of the field, they may not be intellectually ready to absorb it.

CMSC 212, on the other hand, focuses on very low-level concepts, including program performance measurement, assembly language programming, pointer-based data structures and memory management and Unix process management. Again, some of this material may be better placed in CMSC 311 – assuming that CMSC 131 and CMSC 132 don’t change.

3. Teaching pace. Looking at the lecture schedule for CMSC 131 and CMSC 132, the committee was struck by the raw number of topics and pace at which they must be taught. One committee member who has taught CMSC 131 commented that the quantity of material in that course had increased significantly over the last few years. Given that numerous professors complained to the committee that 400-level students are not sufficiently comfortable with the day-to-day programming needed to succeed at that level, this raises the possibility that some of the current material might be removed in order to give more time/depth to other content.

To reinforce these points, the committee heard comments from some well-respected students that our introductory courses are intellectually overwhelming. While these students are succeeding, they believe that there’s simply too much stuff to learn in too short a time frame, that the connections between different courses are too vague, and that the importance/usefulness of many concepts is not apparent.

5 March 2008 Follow-up

In this section we discuss and analyze some changes that have been made to CMSC 131, 132, 212 and 311.

5.1 CMSC 131

In reviewing the introductory computer science sequence last year, our committee expressed some concerns about CMSC 131. The primary concern was that a remarkably wide variety of advanced material was being introduced, including: exception handling, generics, algorithmic complexity, multiple inheritance, sorting, deep versus shallow array copying, etc. We were worried that at least some of this material was too advanced for a first semester course. While it gives students a quick introduction to important topics, it is not clear that they are intellectually prepared to absorb it, and the time spent on these topics necessarily lessens what can be spent on more basic material. Our committee had received complaints from multiple professors that 400-level students often are not sufficiently comfortable with the day-to-day programming needed to succeed at that level. Further, we received comments from well-respected students that our introductory courses are intellectually overwhelming because there was simply too much material to learn in too short a time frame.

These issues led us to recommend previously that some current CMSC 131 material be removed to give the first semester students more time/depth to master the course core topics. We contacted two of the CMSC 131 course
instructors this year, and asked them about what changes had been made to the course offerings since last year. They indicated that the following topics have been removed from the material covered:

- abstract methods/classes
- protected visibility
- model-view-controller design patterns
- observer design patterns
- most of the material on complexity
- searching and sorting
- iterator interface
- Java clone method

This has made it possible for the class to move more slowly through the remaining material, and has been a big improvement for the students who have come to us with little or no previous programming experience. The instructors believe that the course still moves fairly fast. While a few topics have been added (e.g., for-each loops, more coverage of the Java debugger), these are important introductory concepts that merit the time now being spent on them. Our committee’s sense is that CMSC 131 is now substantially improved and our primary concerns have been dealt with effectively, and we do not have any further recommended changes at this time.

5.2 CMSC 132

In reviewing the introductory computer science sequence last year, our committee expressed some concerns about CMSC 132. As with CMSC 131 our primary concerns related to the volume of advanced material covered during the semester. Examples include: Huffman trees, minimum spanning trees, Java networking support, Java threads, and algorithm complexity.

The committee believed that the complexity of this material might be too much for 2nd semester students. It also believed that, given the quantity of material taught, the time spent on individual topics would necessarily be small. This perception was supported by external data sources as described above in our discussion of CMSC 131.

Based on this analysis, we recommended that some material be dropped from the course. We contacted CMSC 132 course instructors this year, and asked them about what changes had been made to the course offerings since last year. They indicated that the following topics have been removed or trimmed from the material covered:

- Java serialization
- Java synchronization
- UML
- Amortized analysis
- Hash table implementation techniques (open addressing vs. chaining)

They also indicated that they had added some material that was dropped by CMSC 131. This material included:

- Java abstract classes
- Big O notation

The instructors tell us that they are spending more time giving detailed feedback on programming exercises – there are 8 during the semester. They generally believe that the class is intensive, but appropriate for the student population.

Our committee’s sense is that CMSC 132 could still use some paring down in terms of material taught, especially since it has picked up some material from CMSC 131. At the same time, the current changes have improved the course. We understand for speaking with the course’s instructors that further improvements are being made. We, therefore, believe the course should continue to be monitored to streamline content delivery.
5.3 CMSC 212 and 311

Our original report identified several areas of possible overlap in the topics covered by these two classes. It also identified some areas where the order in which topics were presented might be improved. To evaluate these issues some recent instructors of these two courses met to compare content and projects.

In some cases they found that alleged overlaps did not actually exist, they were necessary for review purposes, or that they served to deepen exposure to a specific topic. For example:

- The whole first chapter of the CMSC 311 textbook. This material is taught in 212, but acts as a review in 311.
- The memory management and process management topics appear in both courses, but 311 covers it in more depth than 212 does.
- The implementation of malloc and free. These topics were discussed in more detail when 212 was first created, but now are discussed only conceptually; this basic discussion from 212 allows for a more detailed discussion and actual implementation in 311.
- Computer representations of different types of numeric values are also discussed in both 212 and in 311. There are several different angles and concepts of this so there appears to be very little redundancy so no changes are suggested.

In other areas, however, we found inappropriate overlap and are therefore working to correct them. For example:

- Garbage collection and the methods of recognizing garbage; it was determined that this topic is covered sufficiently and fits with other material in 212 so should stay there and be removed completely from 311.
- Lessons on memory management that present padding and alignment are covered in both classes at about the same level; this is being removed from 311 since there wasn’t anything added beyond the content of how it was being covered in 212.
- Since most of the bit manipulation operations and the bit conceptual levels of programming are introduced in CMSC 212, it seems that the "bit shift lab" of 311 would be more appropriate to the content of 212. The suggestion is to move this as one of the earlier projects in 212 but make it significantly smaller so students can get the concept without the same level of time usage.

Overall, the committee feels that these changes should improve the “efficiency” of CMSC 212 and 311. For this course, the committee has no further recommendations at this time.

5.4 CMSC 330

The committee also examined some potential issues related to “orphan concepts” in CMSC 330. After examining these concepts, we decided that they were mostly an artifact of our analysis strategy. That is, the concepts don’t directly feed any 400-level courses, but are nonetheless core computing concepts. Example include finite automata and grammars.

For this course, the committee has no further recommendations at this time.

6 Wrap-up

Our initial report documented a variety of issues and concerns regarding our pre-400-level courses. Over the last year a number of localized changes have been made to improve courses. Further localized changes may still be useful, but we expect to see diminishing returns as many of the simple, high payoff improvements have already been made. Further improvements will likely require a broad rethinking of the overall curriculum, supported by precise data collection and analysis.

We note that such data does not currently exist and the lack of it, severely hampered this committee’s ability to identify and understand just what our key problems really are. For example, we had no way to really evaluate
whether and to what degree previous curriculum changes had had a positive or negative educational effect. Instead we relied on people’s anecdotes and perceptions, which their accompanying strengths and weaknesses.

For example, the committee was unable to answer specific questions about: student educational background and its effect on educational success, why some successful students are exiting the major, why average time to graduation is rising, etc.

In the future the committee recommends that we not undertake major curriculum changes without first gathering baseline data and then collecting in-process measurement.

Additionally, better data will help us address deeper structural issues, helping us to structure the curriculum to better serve incoming students with different educational backgrounds, to make our course offerings more competitive with other departments, and to understand whether specific courses and topics do or do not support our overall educational mission. We understand that some efforts are underway to acquire such data and suggest that these efforts should be supported.