Final Report

British Columbia Computing Education Committee

Flexible Pre-Major Analysis Project

December 31, 2009

Project Lead: Dr. Michael Zastre

Table of Contents

Executive Summary

From February 2008 to December 2009, the British Columbia Computing Education Committee (BCCEC) undertook a Flexible Pre-Major (FPM) Analysis project. The project subcommittee consisted of a representative sample of institutions in BC which offer Information Technology and Communication (ITC) programs (i.e., Computer Science, and Information Systems / Information Technology).

A distinguishing feature of this project was our decision to use learning outcomes as the mechanism for describing a Flexible Pre-Major. This is different from some other approaches by different articulation committees. Our decision was that approaches to the delivery of CS and IS/IT programs can differ significantly between institutions. While students emerging after two years from CS programs at different institutions may have achieved the same learning outcomes, the courses themselves do not necessarily easily articulate using the traditional course-by-course transfer model (i.e., 80% overlap in topics between courses may not always exist).

The project committee oversaw the preparation of eight lists of learning outcomes. Four of these are for Computer Science (CS) topic areas, and four are for Information Systems / Information Technology (IS/IT) topics areas. Nearly all lists consist of "summary learning outcomes" and "enabling learning outcomes". Our intent is that an institution will compare their own program's learning outcomes with the FPM's summary learning outcomes, and a suitable degree of matching of outcomes would indicate the institution is eligible to participate in the BCCEC FPM.

At its Fall 2009 meeting, members of the BCCEC considered the work of the project committee and determined that a CS FPM and an IS/IT FPM are indeed feasible. The project committee therefore recommends the following be considered by participants in the subsequent implementation project, amongst other issues and requirements normally part of a Flexible Pre-Major Implementation project:

- Revise the existing lists of Learning Outcomes (see the Appendix for these lists)
- Obtain institutional consensus on wording of FPM participation.
- Identify other benefits of work from our FPM studies and share it with others interested in this work.

Background and Objectives

1. The BCCEC

The British Columbia Computing Education Committee (BCCEC) is comprised of representatives from Teaching-intensive universities, Research-intensive universities, Community Colleges and Institutes who deliver either Computer Science (CS) programs or Information Systems / Information Technology (IS/IT) programs or both. While the main purpose of committee meetings is to identify and address issues involving articulation and course transfer amongst participating institutions, much time is also devoted to discussions of curricular changes and innovations which are driven to some extent by the steady pace of change in the Information Technology and Communication (ITC) industry.

These changes are many and various: new computer programming languages; new hardware platforms (mobile, embedded, server, desktop); the emergence of ubiquitous networking (wireless, high-speed networks, secure networks); etc. This has not affected all parts of CS and IS/IT curricula, specifically not those with a more mathematical emphasis (i.e. discrete mathematics, algorithmic theory). However, a significant proportion of the curriculum does require constant change, and given an opportunity to implement such changes there is often a desire by an institution's educators to introduce pedagogical innovations that address the learning needs of its students.

Another reason for introducing innovations in course design and delivery has been the "dot-com bust". After experiencing huge increases in student enrolment in the late 1990s resulting in part from the "dot-com boom", student enrolment numbers dropped dramatically in the period from 2002 to 2007. (This was a common experience across North America.) For some BC institutions, the drop in student numbers coincided with the arrival of new infrastructure and resources that assumed continuous student growth based on late-1990s trends, along with expectations that certain targets (e.g., numbers of graduating students) would be met. For other BC institutions, the drop resulted in program cutbacks and, in some cases, program cancellations. In nearly all CS and IS/IT programs there existed pressure to find a way to increase student enrolment while ensuring the degree of student mastery of established learning outcomes was maintained. Teaching and learning practices yielding high withdraw-drop-fail (WDF) rates were no longer considered acceptable, and innovations were introduced as a way to help address the needs of many learners who have an interest and aptitude for the discipline, but for whom previous pedagogical approaches resulted in failure to complete a course of study. Although student numbers are now increasing – albeit nowhere near the levels seen during the "dot-com boom" – the impulse to innovate as a way of improving student learning is ongoing and continuing at many institutions.

Therefore a goal of the committee is to help institutions find ways in which they can introduce the innovations they believe necessary while at the same time ensuring students in those programs do not experience barriers to transfer resulting from such innovations. These barriers would be caused by the significantly different approaches to CS and IS/IT teaching and learning that have been chosen by institutions, along with challenges of establishing and maintaining course-by-course transfer.

2. Project Origins

A recurring topic at BCCEC meetings over the past several years has been the challenge of facilitating articulation of courses between institutions that differ in their approach to first-year instruction. The existence of these differing approaches is not unique to British Columbia. For example, the ACM/IEEE Curriculum Proposal for Computer Science [CS2001] describes six different approaches to even the first computing course ("Imperative First", "Objects First", "Functional First", "Breadth First", "Algorithms First", "Hardware First"). The task force authoring this curriculum proposal tried to whittle the six approaches down to a single approach, but was unable to do so:

> Throughout the history of computer science education, the structure of the introductory computer science course has been the subject of intense debate. Many strategies have been proposed over the years, most of which have strong proponents and equally strong detractors… In the interest of promoting peace among the warring factions, the CC2001 Task Force has chosen not recommend any single approach… Given the current state of the art in this area, we are convinced no one-size-fits-all approach will succeed at all institutions. [CC2001, p. 22]

At our articulation meetings there has been a sense that some courses have articulated successfully due to the goodwill amongst personalities in the committee despite significant differences between the courses. Worded differently, some transfer evaluators not connected with the committee might compare learning outcomes and topics listed in an "Objects First" course description with those from a "Functional First" course description and conclude that they were not the same and should not transfer. That such courses do currently articulate is the consequence of tacit knowledge held by computing education professionals participating in the BCCEC.

This project has therefore originated from a desire of the committee to make explicit this tacit knowledge, and to express it in a form that will support articulation activities. A variety of *ad hoc* approaches have been explored at meetings, none leading to success. Our BCCAT System Liaison Person, Neil Coburn, observed that a more extended study of the problem with an eye to a possible solution framework – in the form of a Flexible Pre-Major (FPM) – might be considered a suitable BCCAT Transfer Innovations project. On May 4, 2007 the BCCEC voted to support the submission of a project proposal for a Flexible Pre-Major Analysis (which we in the BCCEC have termed the "Feasibility Study"). A proposal was subsequently submitted to the Transfer Innovations Fund, and the project was approved. Work on the project began in February 2008.

3. Discipline transfer patterns

The BCCEC has observed three main transfer patterns:

a. Students at teaching universities and colleges within university-transfer programs transfer at some point after one or more years of studies. This is the traditional pattern of transfer.
b. Students completing certificates and diplomas at colleges and teaching universities may wish to transfer to a teaching university (if already at a college) or a research university (if at a college or teaching university) and wish to maximize transfer credit available from studies pursued in the program they have completed.
c. Students at research universities and teaching universities who wish to continue studies at an institution closer to home, usually a college or some other teaching university.

We have not completed a statistical survey of student movement in our discipline. However, such a survey of flows between specific institutions could help us determine which institutions will see the most benefit from an FPM.

4. Project Objectives

There were two primary objectives/outcomes for the project as set out in our original proposal.

The first objective was to determine if a Flexible Pre-Major could be constructed on the assumption that students achieve similar learning outcomes after two years of instruction whether in the transfer program of a sending institution or within the lower-level of a receiving institution's program (and assuming, of course, that the institutions participate in the Computing Education FPM). If a list of such outcomes could be produced, and if it would be feasible for an institution to compare its courses with outcomes on this list, then we would deem an FPM to be feasible. As with other FPMs in the province (such as Music and Sociology & Anthropology) the focus here is on discipline-specific material; differences amongst institutions in their handling of English requirements, complementary studies, SFU's "WQB" requirement, science credit, etc. would not be addressed by our proposal.

The second objective was to prepare the details for a process by which the FPM could be administered. This would include a mechanism used by institutions to join an FPM. Also needed would be a method for the BCCEC to periodically review the description of the FPM (i.e., outcomes to be added, or removed, or modified, etc.)

Another objective not made explicit in the project proposal was that we would investigate the feasibility of two separate FPMs – one for CS programs, and one for IS/IT programs. Despite some overlap in material (e.g., introductory programming), and despite the fact the same institution may offer both programs and even have instructors teaching courses in both programs, these two kinds of Computing Education have enough differences with respect to student learning outcomes that two groups of learning outcomes would be needed.

With respect to our first project objective, we understood that the decision to use learning outcomes would result in much work. However, we were still surprised by the scale of the task, i.e., time and consultation needed to achieve meaningful consensus. For the past 16 months we have benefited from guidance given by Jennifer Orum at BCCAT, and given her experience of other FPM projects she was able to assure us that we were, indeed, making progress on this project objective. This report is therefore primarily a description of our work towards this first objective.

As for the second project objective, this report's "Recommendations" section lists ideas that emerged during discussions by project-group members and within BCCEC meetings. At present we do not yet have language or a process that could be used to administer a Computing Education FPM, and have decided to pass along this task to the future implementation project.

5. Project Team

We attempted to ensure the project team was an accurate representation of the institutions making up the BC transfer system. Participating institutions (and individuals from the institution) were:

- **British Columbia Institute of Technology** (BCIT): Brian Pidcock
- **Langara College**: Bryan Green, Mingwu Chen
- **Okanagan College**: Rick Gee
- **Simon Fraser University**, Burnaby Campus (SFU): Diana Cukierman
- **Thompson Rivers University**: Wayne Babinchuck, Mahnhoon Lee
- **University of British Columbia**, Vancouver Campus (UBC): Donald Acton, Ed Knorr
- **University of the Fraser Valley** (UFV): Ora Steyn
- **University of Northern British Columbia** (UNBC): David Casperson
- **University of Victoria** (UVic): Michael Zastre

One notable absence from the list is the mention of a private institution. As part of any future project proposal regarding a possible BCCEC FPM, we will ensure that at least one such institution (for example, Alexander College or Coquitlam College) is invited to participate.

The team lead for the project was Michael Zastre.

Problem Statement

1. Traditional Articulation, and Rationale for an FPM

Our standard practice in the BCCEC is to use recommendations from BCCAT on course-by-course transfer. If a sending institution's course shares at least 80% of the same material with a receiving institution's course (i.e., learning outcomes are largely identical) then those two courses are said to be equivalent. There are somewhat more complex schemes for which pairs of courses from a sender are said to be equivalent to two courses at a receiver. Central to all of such schemes is the existence of an agreement on transfer between the two institutions that is facilitated by BCCAT.

In Figure 1, for example, institutions A and B cover the same topics in their courses 1, 2 and 3. If institution C wishes to articulate its courses, it may need to establish similar three-by-three articulations of its courses with both A and B. This system does not scale well. One approach to reducing the complexity is to establish block-transfer agreements between institutions. Agreements are still between institutions, and programs are still compared by comparing courses-by-course, with two-by-two or three-by-three schemes considered anomalous.

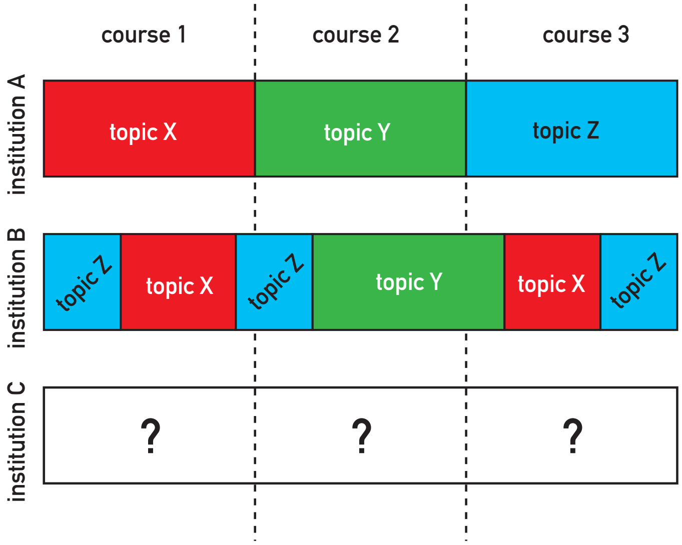

Figure 1: Visualization of mapping topics to courses across institutions

What we have discovered in Computing Education, however, is that two-by-two or even three-by-three articulation for courses are not enough to cover variants in instructional delivery across all institutions. At the same time we recognize that after two years of study, our students appear to have accomplished the same learning outcomes, albeit in different topic order. Therefore we asked ourselves these questions:

- Can we identify the core *discipline-specific* material in the first two years of a CS or IS/IT program?
- Can we then *compare programs against this core* rather than against another institution's programs?
- What would this *comparison system* look like? (Figure 2)

We tried to answer the last question by asking ourselves which level of "granularity" or "detail" should be use to compare courses against each other. The BCCEC found that using "course topics" at any level of detail does not help – it is hard, for example, to get any two instructors to agree what topics such as "recursion" or even "sorting" mean with respect to what is taught in a first- or second-year course. Equivalently it is easy for instructors to agree that certain topics should be included, but only up to the point before discussion turns to how that topic is tested; at this point, disagreement usually is the result.

The point here is not that we want to avoid disagreement. *Our intent is that the mechanism chosen to compare programs should lead to either meaningful agreement, or lead to disagreement that is also meaningful*. "Meaningful disagreement" here indicates that differences in student learning for some items are indeed different between institutions and that there is a reason for such a difference. The best mechanism we found to accomplish this is through the use of *learning outcomes*, and we therefore chose this as the level at which we would describe the content of a Flexible Pre-Major program.

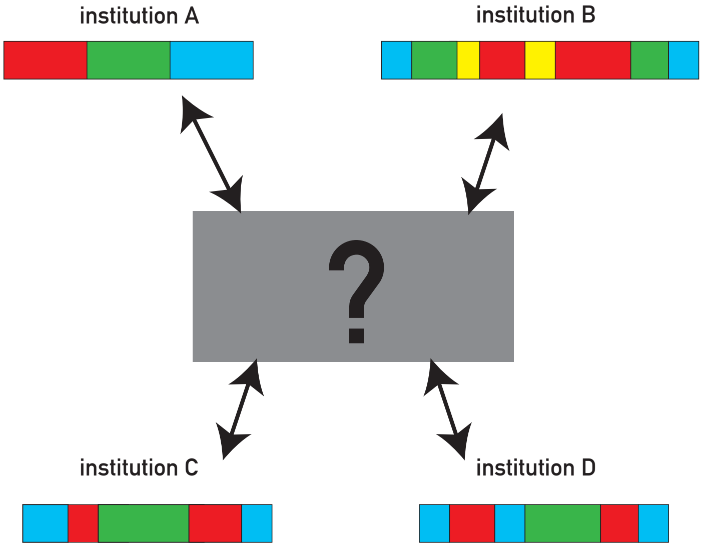

Figure 2: Comparing institutions' programs via some mechanism (here unknown)

An important consequence of using such outcomes to describe an FPM – and potentially identifying an institution's program as matching the FPM – is that a sending institution could consider implementing innovations in their curriculum that can articulate to other institutions without becoming overly worried by course-by-course articulation. This is distinctly different from the current environment where educators at a sending institution often feel constrained to develop a course in such a way that the course is always guaranteed to be accepted by a receiver – and this becomes even more difficult when students from the sending institution may go to a range of receivers each of whom has a different kind of course. Similarly, receiving institutions can choose to implement their own innovations in pedagogy without requiring senders to initiate a reassessment of course equivalence. With an FPM, all participating institutions compare themselves against the FPM and not against another institution. Students completing the FPM at a sender can then transfer discipline-specific credit to a receiver (i.e., a "basket of courses" at the sender matching a "basket of courses" at the receiver).

In summary, an FPM based on learning outcomes provides at least three benefits:

- Sending institutions have confidence that they can focus on appropriate pedagogy for their community of learners.
- Receiving institutions have confidence in the breadth of learning brought by transferring students.
- Students have confidence that their choice of home institution does not constrain their choice of where they can complete their degree.

2. Learning Outcomes as part of the Analysis Project

There exists a large literature on learning outcomes (also sometimes referred to as "learning objectives"). The key is these outcomes describe *learning as accomplished by the student*, as distinct from teaching delivered by the instructor. A document published by BCIT's Learning & Teaching Centre provides a good answer to the question "What is a learning outcome?":

> Learning outcomes specify what learners' new behaviours will be after a learning experience. They state the knowledge, skills, and attitudes that the students will gain through [the] course. Learning outcomes begin with an action verb and describe something observable or measurable. [BCIT2003, p. 2]

Appropriately worded learning outcomes suggest techniques for delivering material in the classroom or in a lab setting. These also suggest approaches towards evaluating student learning. This results in a focus on "outcomes, not processes" [BCIT2003, p. 8] in such a way that different instructors or institutions can choose suitable pedagogical techniques for their students yet agree on the kinds of student learning which should occur.

A common practice when creating learning outcomes is to use an action verb identified by Bloom and others in their work on "learning domains" and "taxonomies" [Bloom56]. Each learning domain (e.g., the "cognitive domain") is a taxonomy of behaviours arranged from the most simple and ending with the most complex. For example, in the cognitive domain, behaviours start with "knowledge" (remembering previously learned material), then lead to "comprehension", then "application", "analysis", "synthesis", and end with “evaluation”. Bloom and his colleagues were motivated to develop these taxonomies as part of the work in several Examination Boards. They were trying to develop a rational approach for assessing exams and tests across different institutions and determining possible equivalence.

The largest amount of effort devoted by the project committee to our Analysis project was, by far, devoted to meetings and discussions for identifying learning outcomes for discipline-specific learning in Computer Science or discipline-specific learning in Information Science / Information Technology. It bears repeating that crafting these outcomes meant eliciting disagreement from discussion participants about the meaning of a particular outcome. A particular wording of a learning outcome may have been too vague, or too specific, or too ambiguous. A learning outcome with an advocate at one institution may have had as passionate a detractor from another institution. (More will be said of this under the “Process” section.)

3. A word of caution about Learning Outcomes

At this point, some readers with experience of learning outcomes may already object strongly towards their suitability for designing an FPM.

Objection 1: “Learning outcomes are too vague.” There can be a place for outcomes phrased using more general language, although usually this is when more specific outcomes are associated with the general outcome. However, at times the vagueness is due to an inappropriate choice of action verb. For example, there is a temptation to use the verb “understand” or “know” as an action verb, yet it can be hard to agree on what these mean when assessing student learning. Those experienced with crafting learning outcomes will instead substitute a more specific verb such as “identify”, “define”, “describe” or “demonstrate” [BCIT2003, p. 4]. Vagueness of learning outcomes is evidence that the authors of the outcome were inexperienced. Vagueness is not a property inherent to learning outcomes.

Objection 2: “We have no course that covers all these learning outcomes, nor could we ever have such a course.” While learning outcomes are often used for course development or program approval, a set of such outcomes does not necessarily correspond to a specific course. Throughout our project we had to remind ourselves over and over again that we were focusing on the first two years of a program and not a specific course. Two learning outcomes with a similar object may stand side-by-side in a list of outcomes (e.g., “Define/describe a binary tree” vs. “Present an algorithm that can be used to find the height of a binary tree”) yet the former might be part of a first-year course, and the latter a part of a second-year course. A list of learning outcomes need not correspond to a single course.

Objection 3: “Writing learning outcomes requires too much effort, and it is far easier to compare course content by using topics or textbooks.” Where there is already significant agreement on what constitutes a course (e.g., in some other BCCAT articulation committees), there may indeed be little gained with respect to articulation by devoting a lot of effort to crafting learning outcomes. In our experience the objection with respect to effort has some validity if it applies to an individual writing outcomes on their own, yet it is not true when a group of experienced educators collaborates (preferably in the same space at the same time) on writing learning outcomes. Such efforts translate into increased precision. The greater the precision, the easier it is to identify clearly when a learning outcome is (or is not) part of a course or program.

Process

1. Plan of meetings

The original project proposal listed a mix of in-person meetings with video-conferenced meetings, including meetings scheduled to directly precede regular BCCEC meetings. We soon discovered that in-person meetings were easier to arrange and were the most productive when crafting learning outcomes.

There were three meetings involving only the project committee:

- February 21 & 22, 2008: Held at UBC
- April 30, 2008: Held at UVic
- December 12, 2008: Held at UFV

and the last meeting involved the project committee with attendees at the Fall 2009 BCCEC meeting:

- October 22 & 23, 2009: Held at UFV

2. Sources and Resources

At its first meeting our project committee identified four topic areas in Computer Science ("Introduction to Programming", "Computer Architecture", "Algorithms and Discrete Structures", "Software Engineering") and four topics areas in Information Systems / Information Technology ("Hardware Systems", "Web Learning", "Information Management", "Networks"). At that meeting we also identified learning outcomes for one CS topic area "Introduction to Programming" (see Appendix A). We identified two main sources for wording of outcomes.

One source is the set of ACM/IEEE Curriculum proposals [CS2001, CS2008, IT2008]. The task force was made up of members of the Association for Computing Machinery and the IEEE Computer Society. A full four-year program is described in [CS2001], and specific IS/IT curricula and possible courses are listed in [IT2008]. Some topic areas described in the curriculum proposal include clearly-written learning outcomes, and these were helpful to our own FPM project. We have tried where possible to indicate when we have used an outcome from these documents.

Another source of outcomes are course outlines and course descriptions from BC institutions. At our second meeting (April 2008) we assigned two or three institutions from the project committee to each of the eight topic areas. Each institution then assigned to a topic area a selection of their courses appropriate to that area (i.e., one course from their first or second year) and prepared a list of learning outcomes based on this course. All learning outcomes for the same topic from different institutions were then combined together, and these combined sets for each topic became the starting point for further discussions by the project committee.

At our third meeting (December 2008) we tackled another topic area ("Algorithms and Discrete Structures") and by the end of the daylong session we completed a list of this area's outcomes (see Appendix B). We also concluded that effort corresponding to a daylong meeting was required for *each remaining* topic area (i.e., six more days of meetings) if we wanted to prepare accurate learning outcomes. We then recognized that one very valuable resource available to the project was the membership of the BCCEC itself. At the May 2009 meeting of the BCCEC we asked it to devote the Fall 2009 meeting to the work required for completing all remaining topic areas. Members of the project committee were prepared to act as facilitators for this work. The BCCEC membership agreed, and the days of October 22 & 23, 2009 were devoted to the other six topics areas.

Findings

1. Learning outcomes (LOs): Enabling vs. Summary

The project committee together with members of the BCCEC has prepared eight lists of learning outcomes, one for each topic area (four in CS, four in IS/IT). These lists appear in the appendix.

We also discovered that a list could be broken into two parts. One part contains "enabling learning outcomes", where each outcome could easily appear as part of a course description or as guides for instructional development. Such outcomes could also guide an educator in crafting an exam question, assignment, project or some other instrument suitable for evaluating student progress. The second part of the list consists of "summary learning outcomes" which are sets of enabling LOs. Each summary LO is still phrased using best practices (e.g., beginning with an action verb) and students achieving such an outcome will have demonstrated mastery of most of the associated enabling LOs.

Another feature of the relationship between these two parts of an LO list is that an enabling LO may belong to two or more summary LOs. Strict partitioning of enabling LOs in summary LO sets is not necessary.

An assumption of the project committee is that an institution will compare its program with the FPM at the level of summary LOs. If educators or administrators are unclear as to the precise meaning of a particular summary LO, they then can refer to the detail available in its associated set of enabling LOs.

Once the resulting course matrix representing the FPM is completed (i.e., assuming the BCCEC goes ahead with an implementation project), the matrix provided to BCCAT will associate a *basket of courses* at institution A with a *basket of courses* at institution B. (This assumes institutions A and B are part of the FPM.)

2. Possible mechanisms for administration of the FPM

One topic to which the project committee turned repeatedely was the question of how the BCCEC can determine when an institution's program can be part of the FPM. At its core our FPM is based on learning outcomes, yet the sets of LOs (summary and enabling) are intended to be a union (i.e., a superset) of LOs from all institutions. Therefore by definition it would be very difficult, if not impossible, for any institution to match 100% of the BCCEC FPM learning outcomes for a given topic.

Therefore an open question is the percentage match an institution's learning outcomes must have with the FPM to be considered sufficient for the institution's participation in the FPM. There are several ways this might be applied. For example, joining the FPM for Computer Science might mean one of:

- 80% match of summary LOs in each of the four topic areas, or

- 80% match of summary LOs from the combination of all four topic areas, or
- no less than an 80% match of summary LOs for any of the four topic areas

where “80%” may be substituted by some other figure decided upon by the BCCEC.

Given that this is a relatively new model for articulation, we expect any implementation phase following this analysis project will begin with consulting Registrars’ Offices from several different institutions.

As for maintenance of the lists of LOs, the level of detail they provide suggests a two-phase approach to review of an FPM, assuming reviews occur every three or four years. In the first phase the list of outcomes are re-examined, with some LOs rephrased, some LOs deleted, some new LOs added, but the majority unchanged. In the year following this review, institutions participating in the FPM can compare their programs against the revised lists. This may mean some institutions substitute different courses against the FPM, or perhaps even revise existing courses to add missing outcomes to their programs.

Changes to the topic areas themselves (i.e., addition of relevant new areas, deletion of obsolete older areas) should occur far less frequently than regular reviews. Such changes could occur on an *ad hoc* basis, and would happen when the BCCEC determines via a formal vote that a topic change is necessary.

3. Several other open questions

What might an FPM look like from a student’s point of view? This refers to the documentation issued by a student’s home institution, and which the receiving institution would accept. Ideally the student’s transcript would have a notation indicating that in completing a course of study they have also completed the student learning comprising a CS or IS/IT Flexible Pre-Major. However, introducing notations to transcripts is difficult and Registrars are rightly hesitant to do so. Another model is that the home institution’s department prepares a letter for the student indicating completion of FPM material, and the participating receiving institution’s department then accepts the letter.

What would be the relationship of the FPM to existing course-by-course transfer agreements? We mentioned earlier some of the challenges in our discipline caused by differing approaches to introductory CS education. If we assume that the flexibility to innovate provided through the FPM is embraced by an institution, then some of that institution’s existing course-by-course transfers may become more and more difficult to maintain. Students completing the “basket of courses” at such an institution will have no difficulty here, but those students completing some but not all courses in such a basket may find they receive general-level credit instead of specific course transfers. This is not a desirable outcome.

How best can such an FPM by communicated to students, faculty and administrators? One of the biggest challenges we found as a project committee was when any one of us forgot to focus on LOs at the program level. We therefore needed to remind ourselves over and over again about our proper focus. One reason why we forget is because we usually think in terms of courses. Current students, faculty colleagues, and administrators handling student transfers are also likely to think in terms of courses when trying to comprehend the FPM (and may therefore draw incorrect conclusions).

Recommendations

At the end of the Fall 2009 BCCEC meeting, the committee considered the work of the project committee and discussed much of the material and questions which are now assembled in this report. The outcome of this discussion was that the BCCEC believed an FPM is indeed feasible, and that it should proceed to an implementation phase. On the last day of the meeting (October 23, 2009) the BCCEC considered the following motion:

> **BCCEC recommends that a funding proposal to the TAC for Phase 2 of the FPM by created for the May [2010] BCCEC meeting.**

"Phase 2" here refers to a "Flexible Pre-Major Implementation Project." The motion was moved, seconded and carried.

What follows are recommendations from the project committee for consideration in the implementation-phase project.

Refine existing LO lists. Not all topics are organized with summary LOs and enabling LOs, and some LOs require rewording (i.e., changing the verb "understand" to a more appropriate verb). Some topics require much more detail, and revising LOs in other topics may result in some contention depending on the form of consultation that is used ("Computer Architecture" is one example). Overall the lists could be made more consistent with each other and may benefit from recent work on taxonomies for learning in ITC (Information Technology and Communication) disciplines. Above all, LOs must be meaningful.

Obtain institutional consensus on wording of FPM participation. The purpose of the FPM lists is to provide flexibility to institutions and to focus on student outcomes. They are not meant to prescribe process or "time on task" (e.g., nothing in the FPM refers to hours spent by students in labs). Receiving institutions need assurance that students transferring via an FPM bring *enough* completed outcomes to be considered equivalent with other students at that institution. Registrars also need some assurance that the administration involved with such transferring students is practicable. Finally, faculty at both sending and receiving institutions should accept the role of such LOs in enabling the transfer of student learning between institutions.

Identify other benefits of work from our FPM studies and share it with others. Some of the extra benefits are:

- Using the LOs to help with interprovincial student transfer.
- Ensuring lists are available to help students discover what they will learn in an FPM program.
- Showing employers these lists so that they understand what CS and IS/IT students learn at our institutions.

- Providing material for dialogue with teachers in the K-12 school system as they already make extensive use of LOs in their own instructional development and delivery.
- Assisting academic units preparing for External Reviews or visits from accreditation bodies.
- Providing a resource for private institutions considering the creation of new ITC programs.

Consider how best to communicate the nature of our FPM to students. There was some discussion by the project committee on the suitability of a website that helps students plan their CS or IS/IT education via learning outcomes.

Some additional comments

At BCCAT's November 6, 2009 JAM (Joint Annual Meeting of Articulation Committee Chairs, SLPs and ICPs), the project committee's work was presented. As part of that presentation some "cautions" (i.e., "lessons learned") for a project such as ours were shared. They are repeated here in the hope they will help any other articulation committees wishing to try an approach such as ours.

Accept no substitute for in-person meetings. As befits information technologists such as ourselves, we tried using e-mail, wikis, distributed-meeting tools, etc. to facilitate our work. In the end, however, we found that crafting meaningful LOs required a lot of give-and-take (not to mention reading body language) in front of a whiteboard. The meanings of certain word, verbs, or technical terms for topics are suitably clarified with such discussion, especially if participants feel comfortable to express doubt, confusion or outright disagreement. An assumption here is that relationships amongst committee members are already collegial such that it is possible to have disagreement without disrespect.

Budget more than a year to do the analysis. Our FPM Analysis project is now complete, and this is 21 months after we started. Our intention in the original project proposal was to perform all work (consultation, deriving lists of LOs, agreement on wording of FPM participation) in 12 months. Thankfully we received much flexibility from BCCAT in the adjustment of project deadlines. We were far too ambitious.

Avoid searching for perfect sets of learning outcomes. There exists no perfect taxonomy of learning. At present the community engaged in research in Scholarship of Teaching and Learning (SoTL) continue to produce additional taxonomies and methods for expressing learning outcomes. We expect our lists to be revised in the future, and also expect this revision to improve the utility of the outcomes. Similarly we would recommend against phrasing learning outcomes solely to ensure there is no disagreement – sometimes we found it helpful to include outcomes in apparent conflict if only because educators adopt certain outcomes due to personal temperament and learning style. We prefer to maintain this kind of diversity, and we achieved this by building consensus.

This is not about courses or topics. There will always exist the temptation to slip back into thinking about course-by-course transfers, and usually these transfers are established by examining lists of topics. In our experience, educators often agree (trivially) on choice of topics, yet disagree about learning outcomes. We strongly recommend setting aside topics and banishing the verb "understand" from LOs.

Acknowledgements

The project committee wishes to thank:

- **Donald Acton, University of British Columbia, Vancouver** for establishing and maintaining an excellent TWiki site that acted as a superb information repository for the project;
- **Neil Coburn, Selkirk College** (and the BCCEC's System Liaison Person) for suggesting the idea of preparing an FPM Analysis project proposal;
- **Jennifer Orum, BCCAT** for advice and encouragement to the BCCEC and her assurances that we were on the right track;
- **Jean Karlinski, BCCAT** for handling the reimbursement of project expenses so quickly and so cheerfully;
- and **Ora Steyn and Paul Franklin, University of the Fraser Valley** for acting as cheerful hosts to the project committee and BCCEC on more than one occasion.

Reference Works

[BCIT2003] "Writing Learning Outcomes." Learning Resources Unit of the British Columbia Institute of Technology. 1996, revised 2003. Available at: https://helpdesk.bcit.ca/fsr/teach/courseprep/htoutcomes.pdf

[Bloom56] "Taxonomy of Educational Objectives: The Classification of Education goals (Handbook 1, Cognitive Domain)", Benjamin Bloom, Editor. Longmans, Green (New York, Toronto). 1956.

[CS2001] "Computing Curricula 2001, Final Report." The Joint Task Force on Computing Curricula, Institute for Electrical and Electronic Engineers (IEEE) Computer Society and the Association for Computing Machinery (ACM). December 2001. Available at: http://www.acm.org/education/education/education/curric_vols/cc2001.pdf

[CS2008] "Computer Science Curriculum 2008: An Interim Revision of CS 2001." Association for Computing Machinery and the Institute for Electrical and Electronic Engineers (IEEE) Computer Society. December 2008. Available at: http://www.acm.org//education/curricula/ComputerScience2008.pdf

[IT2008] "Information Technology 2008: Curriculum Guidelines for Undergraduate Degree Programs in Information Technology." Association for Computing Machinery and the IEEE Computer Society. November 2008. Available at: http://www.acm.org//education/curricula/IT2008%20Curriculum.pdf

Appendices

Eight sets of lists are included here. Listed below for each topic area are the names of those who participated in the discussions that generated each list. Names in italics correspond to those who prepared a written version of the discussion. All lists were converted into a canonical form by the team lead, Michael Zastre (i.e., summary outcomes are itemized by letter, enabling outcomes are itemized by number).

CS: Introductory Programming Languages

- Mingwu Chen, Langara College
- David Casperson, University of Northern British Columbia
- Diana Cukierman, Simon Fraser University, Burnaby
- Rick Gee, Okanagan College
- Bryan Green, Langara College
- *Ed Knorr, University of British Columbia Vancouver*
- Mahnhoon Lee, Thompson Rivers University
- Brian Pidcock, British Columbia Institute of Technology
- Ora Steyn, University of the Fraser Valley
- Michael Zastre, University of Victoria

CS: Algorithms and Discrete Structures

- Donald Acton, University of British Columbia Vancouver
- Wayne Babinkchuk, Thompson Rivers University
- Mingwu Chen, Langara College
- Diana Cukierman, Simon Fraser University Burnaby
- Rick Gee, Okanagan College
- Bryan Green, Langara College
- *Ed Knorr, University of British Columbia Vancouver*
- Ora Steyn, University of the Fraser Valley
- Michael Zastre, University of Victoria

CS: Computer Architecture

- David Casperson, University of Northern British Columbia
- Diana Cukierman, Simon Fraser University Burnaby
- Ed Knorr, University of British Columbia Vancouver
- *Anne Lavergne, Simon Fraser University Burnaby*
- Nalin Wijesinghe, Langara College
- Saif Zahir, University of Northern British Columbia

CS: Software Engineering

- David Casperson, University of Northern British Columbia
- Rick Gee, Okanagan College
- Ed Knorr, University of British Columbia Vancouver
- Anne Lavergne, Simon Fraser University Burnaby
- *Tim Topper, Yukon College*
- Saif Zahir, University of Northern British Columbia

IS/IT: Hardware Systems

- Mohd Abdullah, Thompson Rivers University
- Ken Chan, Columbia College
- Diana Cukierman, Simon Fraser University Burnaby
- *Paul Franklin, University of the Fraser Valley*
- Brian Pidcock, British Columbia Institute of Technology

IS/IT: Web Learning

- Jim Bailey, College of the Rockies
- *Bryan Green, Langara College*
- *Ora Steyn, University of the Fraser Valley*
- Easwari Thoreraj, Selkirk College
- George Tsiknis, University of British Columbia Vancouver

IS/IT: Information Management

- Jim Bailey, College of the Rockies
- Ken Chan, Columbia College
- *Paul Franklin, University of the Fraser Valley*
- Rick Gee, Okanagan College
- Ora Steyn, University of the Fraser Valley

IS/IT: Networking

- Mohd Abdullah, Thompson Rivers University
- Bryan Green, Langara College
- *Brian Pidcock, British Columbia Institute of Technology*
- Easwari Thoreraj, Selkirk College
- Tim Topper, Yukon College
- Raymond Yu, Douglas College

Introduction to Programming Languages (Computer Science)
Enabling Learning Objectives
Prepared: February 21/22, 2008 (@ UBC Vancouver)

Original Source key:
FPM: Flexible Pre-Major Feasibility committee
PFn: ACM/IEEE "Programming Fundamentals" learning outcomes (section n)
PLn: ACM/IEEE "Programming Languages" learning outcomes (section n)
Aln: ACM/IEEE "Algorithms and Complexity" learning outcomes (section n)

#	Enabling Learning Objective	Original source
1	Explain the behavior of a provided code fragment.	FPM
2	Modify an existing code fragment in order to add to its behavior or change its behavior.	FPM
3	Modify and expand short programs that use standard conditional structures and functions.	PF1
4	Modify and expand short programs that use standard iterative control structures and functions.	PF1
5	Write well-structured, well-documented, understandable code.	FPM
6	Describe the role of documentation and comments.	FPM
7	Use language-appropriate idioms.	FPM
8	Write well-structured external documentation.	FPM
9	Design, implement, test, and debug a program that uses each of the followig fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.	PF1
10	Describe the conditional structures available in a language.	FPM PF1
11	Use the appropriate conditional structures available in a language.	FPM PF1
12	Choose appropriate conditional and iteration constructs for a given programming task.	PF1
13	Use pseudocode or diagrams or both to describe the steps involved in solving simple problems.	PF2, FPM
14	Describe the iteration structures available in a language.	FPM, PF1

15	Use the appropriate iteration structures available in a language.	FPM, PF1
16	Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.	PF1
17	Understand the role of pseudocoding and diagramming in decomposing problems.	FPM
18	Demonstrate the role of formal and actual parameters and function arguments.	FPM
19	Identify the necessary properties of good algorithms.	PF2
20	Define algorithm.	FPM
21	Create algorithms for solving simple problems.	PF2
22	Trace the execution of a program (e.g., desk checking).	FPM
23	Describe strategies that are useful in removing errors (debugging).	PF2
24	Use strategies that are useful in removing errors.	FPM
25	Interpret error messages (compiler, run-time, etc.) and understand their causes.	FPM
26	Interpret system documentation.	FPM
27	Interpret language documentation for exploring language features.	FPM
28	Discuss the representation and use of primitive data types and built-in data structures (e.g., strings, arrays, files, etc.).	PF3
29	Describe how strings are allocated, manipulated and used.	PF3
30	Describe how arrays are allocated, manipulated and used.	PF3
31	Describe how records are allocated, manipulated and used.	PF3
32	Describe how lists, stacks and queues are allocated, manipulated and used.	PF3
33	Describe how trees are allocated, manipulated and used.	PF3
34	Describe how graphs are allocated, manipulated and used.	PF3
35	Describe how hash tables are allocated, manipulated and used.	PF3
36	Implement user-defined data structures in a high-level language.	PF3
37	Compare alternative implementations of data structures with respect to performance.	PF3

38	Compare and contrast the costs and benefits of dynamic and static data structure implementations.	PF3
39	Choose the appropriate data structure for modeling a given problem.	PF3
40	Implement user-defined data structures using pointers and references.	FPM
41	Use generic data structures or templates to solve a given problem.	FPM
42	Demonstrate familiarity with contents of industry-standard data structure libraries.	FPM
43	Create a complete suite of tests for a piece of software.	FPM
44	Criticize an existing suite of tests for a piece of software.	FPM
45	Devise appropriate pre- and post-conditions for methods or functions.	FPM
46	Describe the concept of recursion and give examples of its use.	PF4
47	Identify the base case and the general case of a recursively defined problem.	PF4
48	Compare iterative and recursive solutions for elementry problems such as factorial.	PF4
49	Compare and constrast mathematical induction and recursion.	
50	Formulate loop invariants for simple loops.	FPM
51	Demonstrate code correctness given a loop invariant.	FPM
52	Demonstrate loop termination.	FPM
53	Demonstrate correct handling of boundary conditions.	FPM
54	Describe the divide-and-conquer approach.	PF4
55	Implement, test, and debug simple recursive functions and procedures.	PF4
56	Describe how recursion can be implemented using a stack.	PF4
57	Discuss problems for which backtracking is an appropriate solution.	PF4
58	Determine when a recursive solution is appropriate for a problem.	PF4
59	Develop code that responds to exception conditions raised during execution.	PF5.3

60	Explain the difference between event-driven programming and command-line programming.	PF5
61	Design, code, test and debug simple event-driven programs that respond to user events.	PF5
62	Design, code, test and debug simple multi-threaded programs.	FPM
63	Determine when a multi-threaded solution is appropriate for a problem.	FPM
64	Explain the use of big O, omega, and theta notation to describe the behavior of functions.	AL1
65	Use big O, omega, and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms.	AL2
66	Determine the time and space complexity of simple algorithms.	AL1
67	Relate the complexity class of an algorithm to its scalability.	FPM
68	Describe the kinds of operations we can measure in evaluating the performance of an algorithm.	FPM
69	Rank algorithms by rate of growth.	FPM
70	Compare and contrast best-, worst- and average-case behaviors.	FPM
71	Implement a greedy algorithm to solve an appropriate problem.	AL2
72	Implement the most common quadratic and O (N log N) sorting algorithms.	AL3
73	Design and implement an appropriate hashing function for an application.	AL3
74	Design and implement a collision-resolution algorithm for a hash table.	AL3
75	Discuss the computational efficiency of the principal algorithms for sorting, searching and hashing.	AL3
76	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.	AL3

77	Solve problems using fundamental graph algorithms.	AL3
78	Justify the choice of algorithms for a given problem with reference to algorithm time and space properties.	FPM
79	Design and implement a dynamic programming solution to a problem.	AL8
80	Discuss the concept of finite state machines.	AL5
81	Discuss the concept of a deterministic finite automata.	FPM
82	Explain context-free grammars.	AL5
83	Design a deterministic FSM to accept a simple regular expression.	AL5
84	Explain how some problems have no algorithmic solution.	AL5
85	Provide examples that illustrate the concept of uncomputability.	AL
86	Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance and polymorphism.	PL6
87	Design, implement, test and debug simple programs in an object-oriented programming language.	PL6
88	Describe how the class mechanism supports encapsulation and information hiding.	PL6
89	Design, implement, and test the implementation of is-a relationships among objects using a class hierarchy and inheritance.	PL6
90	Compare and contrast the notions of overloading and overriding methods in an object-oriented language.	PL6
91	Expain the relationship between the static structure of the class and the dynamic structure of the instances of the class.	PL6
92	Describe how iterators access the elements of a container.	PL6
93	Interpret UML class diagrams.	FPM
94	identify the classes implied by a problem.	FPM
95	Create a UML class diagram that associates classes identified in a problem.	FPM
96	Create a UML sequence diagram representing object interaction.	FPM
97	Interpret UML interaction diagrams.	FPM

98	Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.	PL3
99	Describe the phases of program translation from source code to executable code and the files produced by these phases.	PL3
100	Explain the differences between machine-dependent and machine-independent translation and where these differences are evident in the translation process.	PL3

Algorithms + Data Structures (Computer Science)
Enabling Learning Objectives (with cross-reference to Summary Learning Objectives)
Prepared: December 12, 2008 (@ University of the Fraser Valley)

#	Enabling Learning Objective	Original source	Summary cross-reference		
1	Understand how to graph the following functions: c, lg x, x, xlg x, x2, 2x	TRU	G		
2	Demonstrate mathematical literacy (competence, familiarity, ability to use to solve problems) in sets, functions, and mathematical symbols	UBC	G		
3	Apply sets and functions to hashing, complexity analysis, counting, and generally supporting exact problem expression.	UBC	G	J	
4	Communicate effectively through set parlance and notation (e.g., be able to translate general problem into rigorous problem statements throughout the course).	UBC	G		
5	Understand the notion of mapping between sets.	UBC	G		
6	Prove one to one and onto for finite and infinite sets.	UBC	G		
7	Recognize the different classes of functions in terms of their complexity.	UBC	H	I	
8	Understand what is meant by asymptotic behavior.	TRU	G	H	
9	Understand the differences between big O, big Omega, and big Theta.	TRU	G	H	I
10	Understand the time taken to execute programs with Big O values (i.e. complexity classes) listed as c, lg x, x, xlg x, x^2, 2^x	TRU	H		
11	Define which program operations we measure in an algorithm in order to approximate its efficiency (e.g., number of instructions, steps, function calls, comparisons, swaps).	UBC	I		
12	Define "input size" and determine the effect (in terms of performance) that input size has on an algorithm.	UBC	H	J	
13	Give examples of common practical limits of problem size for each complexity class.	UBC	J		
14	Explain the differences between best, worst, and average case analysis.	UBC	C	H	

15	Describe why best-case analysis is rarely relevant and how worst-case analysis may never be encountered in practice.	UBC	J			
16	Understand how a sequential search works.	TRU, SFU	E	F	K	
17	Calculate the Big O value for a sequential search	TRU, SFU	F	L		
18	Compute the worst-case asymptotic complexity of an algorithm (e.g. the worst possible running time based on the size of the input (N)).	UBC	C	H	I	J
19	Examine some modifications to a sequential search designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU	M			
20	Differentiate an abstraction from an implementation.	UBC				
21	Describe list, stack and queue data structures along with their public-interface specifications.	SFU, UBC	M			
22	Demonstrate how dynamic memory management is handled in [an imperative language] (e.g., allocation, deallocation or garbage collection, memory heap, run-time stack).	UBC	M			
23	Gain experience with pointers/references in [an imperative language] and their tradeoffs and risks (dangling pointers, memory leaks).	UBC	M			
24	Implement as ADTs -- using both index-based and reference/pointer techniques -- list, stack and queue data structures.	SFU, UBC	M			
25	State examples of problems that can be solved using stack, queues, and dequeues abstract data types.	UBC	L			
26	Recognize algorithms as being iterative or recursive.	UBC	N	O		
27	Prove that a loop invariant holds for a given code or algorithm example.	UBC	P			
28	Describe the relationship between recursion and induction (e.g. take a recursive code fragment and express it mathematically in order to prove its correctness inductively).	UBC	P			
29	Implement iterative and recursive versions of operations on list, stack and queue data structures, and discuss the impact of the implementation choice.	SFU, UBC	L	M	P	
30	Understand how a binary search works.	TRU, SFU	E	K		
31	Calculate the Big O value for a binary search.	TRU, SFU	I			

32	Examine some modifications to a binary search designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU, SFU	I	M	
33	Understand how a hash search works.	TRU, SFU	K		
34	Calculate the Big O value for a hash search.	TRU, SFU	I		
35	Examine some modifications to a hash search designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU	M		
36	Describe tree, hash-table, heaps and priority-queue data structures along with their public-interface specifications.	SFU	K		
37	Implement and manipulate a heap using an array as the underlying data structure.	UBC	M		
38	Implement as ADTs -- using both index-based and reference/pointer techniques -- tree, hash-table, heaps and priority-queue data structures.	SFU	M		
39	Implement iterative and recursive versions of operations on tree, hash-table, heaps and priority-queue data structures, and discuss the impact of the implementation choice.	SFU, UBC	M		
40	Provide examples of appropriate applications for priority queues and heaps.	UBC	K	L	
41	Provide examples of the types of problems that can benefit from a hash data structure.	UBC	K	L	
42	Compare and contrast open addressing and chaining [for heap data structures].	UBC	L		
43	Evaluate collision resolution policies [for heap data structures].	UBC	L		
44	Describe the conditions under which hashing can degenerate from O(1) expected complexity to O(n).	UBC	E	F	L
45	Identify the types of search problems that do not benefit from hashing (e.g., range searching) and explain why.	UBC			
46	Describe how tail-recursive algorithms can require less space complexity than non-tail recursive algorithms.	UBC	N		
47	Draw a recursion tree and relate the depth to a) the number of recursive calls and b) the size of the runtime stack. Identify and/or produce an example of infinite recursion.	UBC	N		

48	Understand how a bubble sort works.	TRU, SFU	A	
49	Calculate the Big O value for a bubble sort.	TRU, SFU	C	I
50	Examine some modifications to the bubble sort designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU	D	I
51	Implement the bubble-sort algorithm.	SFU	D	
52	Understand how a selection sort works.	TRU, SFU	A	
53	Calculate the Big O value for a selection sort.	TRU, SFU, UBC	C	I
54	Examine some modifications to the selection sort designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU	D	I
55	Implement the selection-sort algorithm.	SFU	D	
56	Understand how a insertion sort works.	TRU, SFU	A	
57	Calculate the Big O value for a selection sort.	TRU, SFU, UBC	C	I
58	Examine some modifications to the insertion sort designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU	D	I
59	Implement the insertion-sort algorithm.	SFU	D	
60	Understand how a merge sort works.	TRU, SFU	A	
61	Calculate the Big O value for a merge sort.	TRU, SFU, UBC	C	I
62	Examine some modifications to the merge sort designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU	D	I
63	Implement the merge-sort algorithm.	SFU	D	
64	Understand how a quicksort works.	TRU, SFU	A	
65	Calculate the Big O value for a quicksort.	TRU, SFU	C	I
66	Examine some modifications to the quicksort designed to enhance its performance, and calculate the Big O value for each enhancement.	TRU	D	I
67	Implement the quicksort algorithm.	SFU	D	
68	Compare and contrast the space requirements for merge sort versus quicksort.	UBC	B	C
69	Describe and apply various sorting algorithms; Compare and contrast their tradeoffs.	UBC, SFU	C	

70	State differences in performance for large datasets versus small datasets on various sorting algorithms.	TRU, SFU	B		
71	Define/describe a binary tree.	UBC	K		
72	Apply basic tree definitions to classification problems.	UBC			
73	Explain why a binary tree is useful in CS.	TRU, SFU	L		
74	Present an algorithm that can be used to find the height of a binary tree.	TRU	M		
75	Discuss the Big O value of the algorithm in E3.	TRU, SFU	I		
76	Discuss tree traversal algorithms - InOrder, PostOrder, PreOrder	TRU, SFU	K		
77	Discuss the Big O values of InOrder, PostOrder and PreOrder traversal algorithms.	TRU			
78	Explain why a binary search tree is useful in CS.	TRU	L		
79	Present common binary-search tree algorithms such as search for data, adding data, deleting data.	TRU, SFU, UBC	M		
80	Discuss the Big O value of common binary-search tree algorithms (search for data, adding data, deleting data).	TRU, SFU			
81	Describe the properties of binary trees, binary search trees, and more general trees; and implement iterative and recursive algorithms for navigating them in [an imperative language].	UBC	K	L	
82	Compare and contrast ordered versus unordered trees in terms of complexity and scope of application.	UBC	I	L	
83	Categorize an algorithm into one of the common complexity classes (e.g. constant, logarithmic, linear, quadratic, etc.).	UBC	I		
84	Given two or more algorithms, rank them in terms of their time and space complexity.	UBC			
85	Compare and contrast [the concepts of] space and time complexity.	UBC	C	H	J
86	Describe the structure, navigation and complexity of an order m B+ tree.	UBC	K	L	
87	Insert and delete elements from a B+ tree.	UBC	K		
88	Explain the relationship among the order of a B+ tree, the number of nodes, and the minimum and maximum capacities of internal and external nodes.	UBC	K		

89	Give examples of the types of problems that B+ trees can solve efficiently.	UBC	K	
90	Compare and contrast B+ trees and hash data structures.	UBC	L	
91	Explain why B+ trees are preferred dynamic data structures in relational database systems.	UBC	L	
92	Discuss the tradeoffs in algorithm performance with respect to space and time complexity. E.g., Compare and contrast the space requirements for a linked list (single, double) versus an array-based implementation.	UBC	I	
93	Given a [program fragment], write a formula which measures the number of steps executed as a function of the size of the input (N).	UBC	I	P
94	Take a loop code fragment and express it mathematically in order to prove its correctness inductively (specifically describing that the induction is on the iteration variable).	UBC	P	
95	In simpler cases, determine the loop invariant.	UBC	P	
96	Apply counting principles to determine the number of arrangements or orderings of discrete objects, with or without repetition, and given various constraints.	UBC	G	
97	Use appropriate mathematical constructs to express a counting problem (e.g. counting passwords with various restrictions placed on the characters within).	UBC	G	
98	Identify problems that can be expressed and solved as a combination of smaller sub problems. When necessary, use decision trees to model more complex counting problems.	UBC	P	
99	Solve problems using combinatorial arguments and algebraic proofs.	UBC		
100	State the relationship among recursion, Pascal's Triangle, and Pascal's Identity.	UBC	P	
101	Define binomial distribution and identify applications.	UBC	G	
102	Model and solve appropriate problems using binomial distribution.	UBC	G	
103	Apply basic probability theory to problem solving, and identify the parallels between probability and counting.	UBC	G	

104	Define various forms of the pigeonhole principle; recognize and solve the specific types of counting and hashing problems to which they apply.	UBC	P
105	Discuss the Big O of spanning-tree algorithms.	TRU	
106	Perform breadth-first and depth-first searches in graphs.	UBC	K
107	xplain why graph traversals are more complicated than tree traversals.	UBC	K
108	Discuss Prim's and Kruskal's minimal spanning-tree algorithms.	TRU	K
109	Discuss the Big O of minimal spanning-tree algorithms.	TRU	
110	Describe the properties and possible applications of various kinds of graphs (e.g., simple, multigraph, bipartite, complete), and the relationships among vertices, edges, and degrees	UBC, TRU	L
111	Prove basic theorems about simple graphs (e.g. handshaking theorem).	UBC	
112	Explain the computer representation of graphs.	TRU	K
113	Convert between adjacency matrices / lists and their corresponding graphs.	UBC	K
114	Determine whether a given graph is a subgraph of another.	UBC	
115	Discuss the complexity of the Travelling Salesman problem	TRU	
116	Explain Dijkstra's Algorithm for the Shortest Path in a graph	TRU	
117	Discuss the Big O of Dijkstra's algorithm	TRU	
118	Apply object oriented and modular design techniques to an application problem to design a software solution.	SFU	
119	Select the most appropriate data structure (lists, stacks, queues, trees, hash tables, heaps, priority queues) for a solution to a problem.	SFU, UBC	L
120	Implement an application design, including an implementation an appropriate data structure (lists, stacks, queues, trees, hash tables, heaps, priority queues).	SFU	M
121	Analyze [imperative-language] programs and functions to determine their algorithmic complexity.	UBC	I

Algorithms + Data Structures (Computer Science)
Summary Learning Objectives (with cross-reference to Enabling Learning Objectives)
Prepared: December 12, 2008 (@ University of the Fraser Valley)

Set of sorting algorithms Ss (in something of a partial-order indicating importance): selection, insertion, bubble, merge, quicksort, heapsort.
Set of data structures Ds (in something of a partial-order indicating importance): stacks, lists, queues, binary trees, binary search trees, hash tables, heaps, priority queues, graphs.

Set of operations Ops on data structures: insertion, deletion, traversal, search.

#	Summary Learning Objective	Enabling cross-reference														
A	Illustrate / trace the operation of sort s from Ss.	48	52	56	60	64										
B	Choose / justify a sort given a specific problem.	68	70													
C	Compare / contrast tradeoffs of sorting algorithms s, t, and u from Ss.	14	18	49	53	57	61	65	68	69	85					
D	Implement (apply) and modify sorting algorithm s from Ss.	50	51	54	55	58	59	62	63	66	67					
E	Illustrate / trace the search operation on data structure d from Dd.	16	30	44												
F	Compare / contrast tradeoffs of search algorithms s and t.	16	17	44												
G	Demonstrate mathematical literacy in sets, functions & mathematical symbols	1	2	3?	4	5	6	8	9	96						
		101*	102*	103* (* = math/stats)					97 (related to a problem)							
H	Given a Big O expression, state what it implies (also: possibly big Omega, big Theta)	7	8	9	10	12	14	18	85							
I	Categorize an algorithm / data-structure operation into common complexity classes {derive function?}	7	9	11	18	31	32	34	49	50	53	54	57	58	61	62
		65	66	75	82	83	92	93	121							
J	Given a specific problem, use {resource requirements | complexity classes} when comparing & contrasting different solutions.	3	12	13	15	18	85									
K	Illustrate / trace / explain operation op from Ops on data structure d from Ds.	16	30	33	36	40	41	71	76	81	86	87	88	89	106	107
		108	112	113												
L	Choose / justify a data structure given a specific program {? using analysis} {? +complexity classes}	17	25	29	40	41	42	43	44	73	78	81	82	86	90	91
		110	119													
M	Implement (apply) operations from Ops on / for data structure d from Ds.	19	21	22	23	24	29	32	35	37	38	39	74	79	120	
N	Illustrate / trace / explain recursive solutions.	26	46	47	+{tree operations}											

<table>
<tr><td>O</td><td>Implement recursive solutions to appropriate problems.</td><td>26</td><td colspan="5">+{tree operations}</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>P</td><td>Prove properties about {algorithms | programs} that use loops and recursion.</td><td>27</td><td>28</td><td>29</td><td>93</td><td>94</td><td>95</td><td>98</td><td>100</td><td>104</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Computer Architecture (Computer Science)
Enabling Learning Objectives
Prepared: October 23, 2009 (@ University of the Fraser Valley)

Original Sources:
List 1: ACM/IEEE 2001 "Architecture and Organization" learning outcomes
List 2: UBC CPSC 213

Other feedback: Opinions on inclusion sought from UBC-V and SFU-B (i.e., institution's mention indicates coverage in years 1 & 2).

#	Enabling Learning Objective	FPM consensus	Other feedback
1	Describe the progression of computer architeture from vacuum tubes to VLSI.	Yes	SFU
2	Demonstrate an understanding of the basic building blocks and their role in the historical development of computer architecture.	Yes	SFU, UBC
3	Use mathematical expressions to describe the functions of simple combinational and sequential circuits.	Yes	SFU, UBC
4	Design a simple circuit using the fundamental building blocks.	Yes	SFU, UBC
5	Explain the reasons for using different formats to represent numerical data.	Yes	SFU, UBC
6	Explain how negative integers are stored in sign-magnitude and twos-complement representation.	Yes	SFU, UBC
7	Convert numerical data from one format to another.	Yes	SFU, UBC
8	Discuss how fixed-length number representations affect accuracy and precision.	Yes	SFU, UBC
9	Describe the internal representation of nonnumeric data.	Yes	SFU, UBC
10	Describe the internal representation of characters, strings, records and arrays.	Yes	SFU, UBC
11	Explain the organization of the classical von Neumann machine and its major functional units.	Yes	SFU, UBC
12	Explain how an instruction is executed in a classical von Neumann machine.	Yes	SFU, UBC

13	Summarize how instructions are represented at both the machine level and in the context of a symbolic assembler.	Yes	SFU, UBC
14	Explain different instruction formats, such as addresses per instruction and variable length vs. fixed-length formats.	Yes	SFU, UBC
15	Write simple assembly language program segments.	Yes	SFU, UBC
16	Demonstrate how fundamental high-level programming constructs are implemented at the machine-language lvel.	Yes	SFU, UBC
17	Explain how subroutine calls are handled at the assembly level.	Yes	SFU, UBC
18	Explain the basic concepts of interrupts and I/O operations.	Yes	SFU, UBC
19	Identify the main types of memory technology.	Yes	SFU, UBC
20	Explain the effect of memory latency on running time.	Yes	SFU, UBC
21	Explain the use of memory hierarchy to reduce the effective memory latency.	Yes	SFU, UBC
22	Describe the principles of memory management.	Yes	SFU, UBC
23	Describe the role of cache and virtual memory.	Yes	SFU
24	Explain the workings of a system with (simple) virtual memory management.	Yes	SFU, UBC
25	Explain how interrupts are used to implement I/O control and data transfers.	unsure	SFU
26	Identify various types of buses in a computer system.	unsure	SFU
27	Describe data access from a magnetic disk drive.	unsure	SFU, UBC
28	Compare the common network configurations.	unsure	
29	Identify interfaces needed for multimedia support.	unsure	
30	Describe the advantages and limitations of RAID architectures.	unsure	
31	Compare alternative implemention of datapaths.	No	SFU
32	Discuss the concept of control points and the generation of control signals using hardwired or microprogrammed implementations.	No	SFU
33	Explain basic instruction-level parallelism using pipelining and the major hazards that may occur.	No	SFU
34	Discuss the concept of parallel processing beyond the classical von Neumann model.	No	

35	Describe alternative architectures such as SIMD, MIMD, and VLIW.	No	
36	Explain the concepts of interconnection networks and characterize different approaches.	No	
37	Discuss the special concerns that multiprocessing systems present with respect to memory management and describe how these are addressed.	No	
38	Describe superscalar architectures and their advantages.	No	
39	Explain the concept of branch prediction and its utility.	No	
40	Characterize the costs and benefits of prefetching.	No	
41	Explain speculative execution and identify the conditions that justify it.	No.	
42	Discuss the performance advantages that multithreading can offer in an architecture along with the factors that make it difficult to derive maximum benefits from this approach.	No	UBC
43	Describe the relevance of scalability to performance.	No	UBC
44	Explain the basic components of network systems and distinguish between LANs and WANs.	unsure	UBC
45	Discuss the architecture issues involved in the design of a layered network protocol.	unsure	UBC
46	Explain how architectures differ in network and distributed systems.	unsure	
47	Discuss architectural issues related to network computing and distributed media.	unsure	

Software Engineering (Computer Science)
Summary Learning Objectives (with cross-reference to Enabling Learning Objectives)
Prepared: October 22, 2009 (@ University of the Fraser Valley)

#	Summary Learning Objective	Enabling cross-reference														
A	Complete a team-based project using appropriate SE tools and technologies (Bloom's: Synthesis)	all														
B	Demonstrate comprehension of software-engineering jargon including techniques and best practices, e.g., refactoring, reusability, product (Bloom's: Knowledge)	1	66	and implicit in most others												
C	Select, with justification, an appropriate set of tools to support the development of a particular software product. (Tools and Environments)	7	8	9	10	11	12	31	32	33	40	42	71	72		
D	Display competence with enabling technologies for software engineering, e.g., "make", OS, IDE, browser (Tools & Environments; Bloom's: Apply)															
E	Demonstrate through involvement in a team project the central elements of team building and team management. (Software Engineering Management)	25	34	35	36	37	44	59								
F	Apply common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system (Requirements).	51	52	53	54	55	56	57	58	59						
G	Create and specify the software design for a medium-sized software project using a software requirement specification (e.g., structured or object-oriented) and appropriate design notation. (Software Design & Quality)	14	15	16	17	18	19	20	21	22	23	24	25	30	43	69
H	Evaluate different designs prepared as solutions to the same problem. (Software Design & Quality)	26	27	28	29											
I	Explain the software life cycle and its phases including the deliverables that are produced. (Software Development Process & Lifecycle)	39	40	41	42											
J	Select, with justification, the software development models and process elements most appropriate for the development and maintenance of a particular software product (Software Development Process & Lifecycle)	43	44													

K	Quickly construct high-quality software to realize a design (Construction)	2	3	4	5	6	7	8	9	10	11	12	13	45		
L	Test code with unit tests, system tests, and user tests. (Testing)	60	61	62	63	64	66	67	68	69	70					
M	Discuss issues arising in software deployment, maintenance and support. (Production -- Deployment, maintenance, support)	30	46	48	49											

Software Engineering (Computer Science)

Enabling Learning Objectives (with cross-reference to Summary Learning Objectives)

Prepared: October 22, 2009 (@ University of the Fraser Valley)

#	Enabling Learning Objective	Original source	Summary cross-reference		
1	Define software engineering, and describe its history.	SFU	K		
2	Implement a software system. (Moved toSummary Objective K)	SFU	K		
3	Implement design through coding. (Moved to Summary Objective K)	BCIT	K		
4	Quickly implement high-quality code from a design. (Moved to Summary Objective K)	UBC	K		
5	Break up implementation work into units for parallel implementation.	UBC	K		
6	Explain several implementation philosophies.	UBC	K	C	D
7	Improve code quality and productivity by using software tools.	UBC	K	C	D
8	Coordinate implementation efforts using a code repository.	UBC	K	C	D
9	List the typical operations provided by an SCM tool.	UVic	K	C	D
10	Describe various aspects of software configuration management.	SFU	K	C	D
11	Justify the use of software-configuration management (SCM) tools such as Subversion, CVS, etc.	UVic	K	C	D
12	Identify and eliminate problems using an issue tracking system.	UBC	K		
13	Design and apply code standards.	SFU	G	H	
14	Explain and apply good design principles.	UBC	G	H	
15	Explain and apply common design patterns.	UBC	G	H	
16	Select and apply appropriate design patterns in the construction of a software application.	BCIT, UBC, ACM/IEEE	G	H	
17	Recognize basic architectures.	UBC	G	H	
18	Design and specify a system's architecture.	SFU	G	H	
19	Design and specify the class-level structure of a software system.	SFU	G	H	
20	Identify the relationships between classes.	BCIT	G	H	

21	Extend the analysis classes to represent the design use cases and identify specific object instances.	BCIT	G	H	
22	Add/modify relationships between classes and objects to further extend the design.	BCIT	G	H	
23	Represent analysis and design models using use case, sequence, collaboration, class, and state machine diagrams.	BCIT	G	H	
24	Design a project with UML.	UBC	G	H	E
25	Design a project in a group setting.	UBC	G	H	
26	Describe the qualities of a good software system and understand their value.	SFU	G	H	
27	Discuss the properties of good software design including the nature and the role of associated documentation.	UVic	G	H	
28	Evaluate the quality of alternative software designs based on key design principles and concepts.	ACM/IEEE	G	H	
29	Measure the size of a project.	(Okanagan)	G	H	M
30	Use feedback from implementation to refine design.	BCIT	C	D	
31	Analyze and evaluate a set of tools in a given area of software development (e.g. management, modeling, or testing).	UVic, ACM/IEEE	C	D	
32	Demonstrate the capability to use a range of software tools in support of the development of a software product of medium size.	UVic	C	D	
33	Utilize tools to manage and support a software development team such as software configuration management tools (version control repositories), project management tool (task schedulers, meetings) and communication tools (email, shared websites, instant messaging).	SFU	E		
34	Explain the process by which they would organize the solution to a medium-sized non-trivial problem involving a group of programmers.	UVic	E		
35	Apply good project management practices to a software project, such as risk analysis, task/resource scheduling, human resource management, and continuous progress monitoring.	SFU	E		
36	Identify and resolve common team-related issues such as communication problems and decision making.	SFU	E		

37	Review and evaluate team member performance.	SFU	I	J		
38	Explain the software life cycle and its phases including the deliverables that are produced. (Moved to Summary Objective I)	UVic, UBC	I	J		
39	For each of several software project scenarios, describe the project's place in the software life cycle, identify the particular tasks that should be performed next, and identify metrics appropriate to those tasks.	UVic	I	J	M	
40	Identify the principal issues associated with software evolution and explain their impact on the software life cycle.	UVic	I	J		
41	Explain the risks of skipping or reducing a phase of the lifecycle.	UBC	C	D	I	J
42	Recognize the types of tools that are used in each phase (of the software life cycle).	UBC	G	H	I	J
43	Compare the traditional waterfall development model to the incremental model, the agile model, the object-oriented model, and other common models.	UVic, SFU	I	J		
44	Apply a software life cycle model of Object-Oriented paradigm, and its methodology to a multi-member software development project.	SFU	I	J	K	
45	Create user documentation.	SFU	M			
46	Discuss the challenges of maintaining software.	UVic	M			
47	Discuss issues arising in software system deployment, maintenance, and support. (Moved to Summary Objective M)	SFU	M			
48	Discuss the challenges of maintaining legacy systems and the need for reverse engineering.	ACM/IEEE	M			
49	Identify weaknesses in a given simple design, and highlight how they can be removed through refactoring.	ACM/IEEE	M			
50	Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system. (Moved to Summary Objective F)	UVic	F			
51	Argue for the need for requirements.	UBC	F			
52	Eliciting, analyzing, specifying, and verifying functional and non-functional requirements.	SFU	F			

53	Describe several types of requirements.	UBC	F				
54	Elicit requirements from a client.	UBC	F				
55	Identify and complete use cases.	BCIT	F				
56	Refine use cases to serve as foundation for design.	BCIT	F				
57	Identify classes based on use cases.	BCIT	F				
58	Recognize good and bad requirements.	UBC	F				
59	Explain the typical difficulties of technical communication.	UBC	E	F			
60	Construct a software test plan.	BCIT, UVic	L				
61	Create, evaluate and justify, and implement a test plan for a medium-size code segment.	UVic, BCIT	L				
62	Distinguish between the different types and levels of testing (unit, integration, systems and acceptance) for medium-size software products.	UVic	L				
63	Create test cases.	BCIT,UVic	L				
64	Undertake, as part of a team activity, an inspection of a medium-size code segment.	UVic	L				
65	Test code with units tests, system tests, and user tests. (Moved to Summary Objective L)	UBC, BCIT, UVic	C	D	L		
66	Explain basic testing terminology.	UBC	L				
67	Recognize common testing frameworks employed in the industry.	UBC	L				
68	Verifying and validating all artifacts created during the process.	SFU	G	H	I	J	L
69	Testing the resulting system through unit testing, integration testing, system testing, etc.	SFU	L				
70	Describing user acceptance testing.	SFU	L				
71	Describe the role that tools can play in the validation of software.	UVic	C	D	L		
72	Write test scripts.	BCIT, UVic	C	D	L		

Hardware (Information Systems / Information Technology)
Enabling Learning Objectives
Prepared: October 23, 2009 (@ University of the Fraser Valley)

Wording for all learning outcomes prepared by working group at UFV meeting.

#	Enabling Learning Objective	Additional Comments
1	Identify the latest trends and development in microcomputers.	*Overall*
2	Describe and classify the components of computers and peripherals.	
3	Describe, classify and identify how the various PC components are connected and they communicate to accomplish different tasks	
4	Describe the construction and operation of the Central Processing Unit in terms of instruction execution.	
5	Explain the structure and operation of hierarchy of memory in terms of program execution.	
6	Explain how machine language provides the foundation for all programming languages.	
7	Compare different CPU and PC architectures.	
8	Install, maintain and troubleshoot basic computer hardware and software in a LAN environment, demonstrating basic problem solving methodologies.	
9	Use the Internet to assist in solving hardware problems and installing software and firmware updates from the Internet.	
10	Format, partition and reorganize a disk.	
11	Describe how disk storage works, and explain the factors that influence performance.	
12	Identify and understand the basic hardware and software necessary to connect the PC to a network.	*Also OS*
13	Explain the need for and the technologies available for backup and restore.	

14	Describe how video cards and monitors work; determine the settings of both; manipulate the parameters affecting performance.	
15	Describe how data is stored to and retrieved from optical disk.	
16	Describe how data is stored to and retrieved from flash RAM devices such as USB memory, SD cards and other solid state memory technologies.	
17	Describe how audio devices work in a personal computer.	
18	Install and configure a Windows Server and networking services.	*OS*

Web Learning (Information Systems / Information Technology)
Summary Learning Objectives (with cross-reference to Enabling Learning Objectives)
Prepared: October 22, 2009 (@ University of the Fraser Valley)

#	Summary Learning Objective	Enabling cross-reference														
A	Given a problem, suggest an internet infrastructure suitable to solve the problem and justify your choice.	1	2	3	4	5	6	7	57	61	62	64	66			
B	Use services for communication and to access internet-based resources.	5	6	8	36	37	41	67								
C	Apply copyright law, ethics and internet law to a website.	8	38	44												
D	Create a dynamic website that incorporates a user friendly design.	16	17	22	23	24	25	26	27	28	29	30	31	32	33	39
		42	43	60	68											
E	Create a dynamic website that incorporates XHTML/CSS.	8	9	10	11	12	13	14	15	16	27	33	34	35	43	55
		56	57	58	60	63										
F	Create a dynamic website that incorporates a scripting language (client- or server-side).	8	39	43	45	46	47	48	49	50	51	52	53	54	55	56
		57	58	59	60	61	62	63	65							
G	Create a dynamic website that incorporates generally accepted standards.	9	13	25	26	42	43	60	61	62						
H	Create a dynamic website that incorporates web standards.	8	42	43	60	67										
I	Create a secure website that accesses a database.	8	43	58	61	62	65									
J	Use an appropriate range of tools to create a multimedia website.	18	19	20	21	22	25	27	33	34	35	37	38	40	41	42
		56	67	68												

Web Learning (Information Systems / Information Technology)
Enabling Learning Objectives (with cross-reference to Summary Learning Objectives)
Prepared: October 22, 2009 (@ University of the Fraser Valley)

Original source key:
LC is for Langara College
OC is for Okanagan College

#	Enabling Learning Objective	Original source	Summary cross-reference						
1	Describe the major hardware and software components and how they are related to the internet infrastructure	LC	A						
2	Describe the roles and importance of TCP/IP in the Internet.	LC	A						
3	Describe how the DNS system works.	LC	A						
4	List several of the different top-level domains and describe their intended audiences.	LC	A						
5	List several application level protocols (e.g., POP, SMTP, FTP, HTTP).	LC	A	B					
6	Provide examples of how and where they get used on the Internet.	LC	A	B					
7	Compare client/server and peer-to-peer architectures.	LC	A						
8	List and explain several of the considerations when a company hosts a web site (e.g., amount of disk space, monthly transfer limits, etc.).	LC	A	B	C	E	F	H	I
9	Create W3C valid XHTML web pages using headings, paragraphs, logical formatting, lists, tables, images, hyperlinks and character entities.	LC	E	G					
10	Differentiate between an absolute and a relative URL and be able to construct the correct one at the correct time.	LC	E						
11	Use the XHTML tags div and span to create sections for styling.	LC	E						
12	Create tag, pseudo-class, class and id selectors using basic properties (e.g., font, color, text-decoration, text-align, background, list-style-type, etc.).	LC	E						
13	Place properly formed CSS rules in an external or internal style sheet or as an inline style.	LC	E	G					

14	Analyse a set of CSS rules for the cascade effect and render the resultant styling to a webpage.	LC	E		
15	Describe the CSS box model.	LC	E		
16	Use the CSS box model and positioning properties to create a web page with either 2 or 3 columns, a mast head and a footer.	LC	D	E	
17	Compare and contrast various web page layouts: liquid, fixed and jello/elastic.	LC	D		
18	Describe the kinds of pictures that are best stored in raster and vector formats.	LC	J		
19	Describe the characteristics of raster and vector formats, including typical filename extensions, amount of transparency permitted and type of compression used.	LC	J		
20	Describe how resolution and pixel depth (8-bit indexed, 24-bit RBG and 8-bit grayscale) affect the appearance of a raster image and its stored size on disk.	LC	J		
21	Use an image manipulation program to perform the following actions: scaling, rotating, cropping, down sampling, repairing an image by erasing an object, removing the background from a raster image, converting between the various file formats, creating a composite using layers and layer masks, and creating simple GIF animations using layers.	LC	J		
22	Describe the RGB, HSV and CMYK color models.	LC	D	J	
23	Select colors based on the color harmonies: monochromatic, complementary, analogous, and triadic.	LC	D		
24	Describe the five basic webpage design principles (contrast, repetition, alignment, proximity, communicability).	LC	D		
25	Design a web site using the five basic webpage design principles.	LC	D	G	J
26	Critique a web site using the five basic webpage design principles.	LC	D	G	
27	Design a small web site for a mobile device taking into account the limited screen resolution, colour depth, bandwidth, and reduced keyboard.	LC	D	E	J

28	Describe the characteristics of exact and ambiguous site organizational schemes, along with their sub-schemes, giving examples of where they are used appropriately.	LC	D					
29	Select an appropriate organizational scheme for a web site.	LC	D					
30	Compare two web sites according to their organizational scheme	LC	D					
31	Describe the various kinds of site organizational structures.	LC	D					
32	Identify on a site the primary and secondary navigation elements (e.g., breadcrumb trails, site maps, and site index).	LC	D					
33	Create a web site with primary and secondary navigation elements.	LC	D	E	J			
34	Use a web authoring tool (eg Dreamweaver) to maintain a website.	LC	E	J				
35	Create and use authoring tool (eg Dreamweaver) templates to create a website.	LC	E	J				
36	Describe what a search engine is and how to provide it queries using AND, OR, NOT and exact phrases.	LC	B					
37	List and use several search engine optimization techniques to improve a website's ranking in the search results.	LC	B	J				
38	Analyse a site's log file to determine when visitors arrive, from where, and which pages they view.	LC	C	J				
39	Describe how cookies can be used in a web site to customize the appearance for return visitors.	LC	D	F				
40	Use an available tool to create a simple two dimensional animation using multiple layers and object tweening.	LC	J					
41	Describe the fundamental characteristics and uses of e-commerce, blogs, wikis, content management systems, and RSS feeds.	LC	B	J				
42	Use an available content management system to create a website	LC	D	G	H	J		
43	Create a secure e-commerce web site using an appropriate existing payment processing service	LC	D	E	F	G	H	I
44	Discuss the legal issues of copyright, trademarks, hate literature, libel, jurisdiction and the web.	LC	C					

45	Use a scripting language to write several programs to solve problems	LC	F				
46	Insert a scripting l'anguage program into an XHTML page.	LC	F				
47	Use built-in operators, variables, literals to create expressions.	LC	F				
48	Learn how to use scalar, array and hash variables in a script.	OC, LC	F				
49	Use [the scripting language's] string manipulating features.	OC	F				
50	Describe the following constructs – selection, repetition, subprograms.	OC, LC	F				
51	Use both selection structures – if, switch.	LC	F				
52	Use all repetition structures – while, for, do.	LC	F				
53	Create a function to solve a problem.	LC	F				
54	Distinguish between void and value returning functions.	LC	F				
55	Use all XHTML form tags (buttons, text, textarea, radio, checkbox, select).	LC	E	F			
56	Describe the Document Object Model (JavaScript? Object Hierarchy) and properties and methods of form elements.	LC	E	F	J		
57	Use events and event handlers to create an interactive web page.	LC	A	E	F		
58	Describe the process of validating and submitting form data.	LC	E	F	I		
59	Save data to a data file and a database.	OC	F				
60	Create a dynamic web site	LC	D	E	F	G	H
61	Explain the role of a CGI script in creating interactive Web sites.	OC	A	F	G	I	
62	Write a small CGI script to dynamically create a web page in response to a request, collect data from a Web page visitor or send an email	LC, OC	A	F	G	I	
63	Use server side includes to dynamically create a web page.	LC	E	F			
64	Describe some of the major historical events in the evolution of the internet.	(added by FPM)	A				
65	Apply the appropriate OS security and permissions to allow a script to execute.	(added by FPM)	F	I			
66	Describe some of the encryption techniques used on the Internet.	(added by FPM)	A				
67	Use SSL tools to create a secure connection.	(added by FPM)	B	H	J		
68	Create a video or audio podcast.	(added by FPM)	D	J			

Information Management (Information Systems / Information Technology)
Summary Learning Objectives (with cross-reference to Enabling Learning Objectives)
Prepared: October 22, 2009 (@ University of the Fraser Valley)

#	Summary Learning Objective	Enabling cross-reference														
A	Correctly use terminology relevant to information management.	1	2	3	4	5										
B	Describe organizational needs relating to data acquisition, use, retention and disposition.	6	7	8	9	10	41	42	43	44						
C	Describe different database models and differentiate between them.	20	21	22	23	24	49									
D	Model a relational database.	30	31	35	36	37										
E	Design a normalized database.	25	26	27	28	29	33	34	35	36						
F	Write syntactically correct and accurate SQL statements.	11	12	13	14	15	16									
G	Embed relational technology in a programming or web environment.	18	19	49	50	51										

Information Management (Information Systems / Information Technology)
Enabling Learning Objectives (with cross-reference to Summary Learning Objectives)
Prepared: October 22, 2009 (@ University of the Fraser Valley)

Original sources of enabling learning outcomes (except where noted in the comments) are:
ACM/IEEE 2001 CS Curriculum Proposal, "Information Management" section
ACM 2008 Curriculum Proposal, "Information Management" section

#	Enabling Learning Objective	Summary cross-reference	Comment
1	Differentiate and use key terms such as: information, data, database, database management system, metadata, data mining.	A	
2	Explain the role of data, information, and databases in organizations.	A	
3	Explain how data storage and retrieval has changed over time.	A	
4	Explain the advantages of a database approach compared to traditional file processing.	A	
5	Identify and explain the general types of databases: personal, workgroup, department, enterprise.	A	
6	Explain how the growth of the Internet and demands for information for users outside the organization (customers and suppliers) impact data handling and processing.	B	
7	Define data quality, accuracy and timeliness, and explain how their absence will impact organizations.	B	
8	Describe mechanisms for data collection and their implications (automated data collection, input forms, sources).	B	
9	Explain basic issues of data retention, including the need for retention, physical storage, security.	B	
10	Explain why data backup is important and how organizations use backup and recovery systems.	B	
11	Formulate and test SQL queries using SELECT FROM WHERE ORDER BY blocks.	F	
12	Recognize the need for logical operators, set operators, UNION, DISTINCT, LIKE, and BETWEEN operators, and use them appropriately.	F	

13	Formulate and test queries using aggregate functions with GROUP BY HAVING clause.	F	
14	Formulate and test queries using use sub-queries, VIEWS and joins in combinations with the options listed above.	F	
15	Format output (header, footer, totals, subtotals etc.) reports using SQL options and post-processing features of environment like SQL*Plus.	F	
16	Declare appropriate data types, sizes and constraints on elements and their combinations including DATE and TIME types, create TABLE/VIEW with SELECT AS, and use INSERT, UPDATE and DELETE options.	F	
17	Demonstrate an understanding of XPath and XQuery.	🟥	*Delete?*
18	Formulate and test queries using query by example.	G	*Suitable?*
19	Use embedded SQL queries.	G	*Suitable?*
20	Give a brief history of database models and their evolution.	C	
21	Describe the features of the relational model including relations, tuples, attributes, domains and operators.	C	
22	Demonstrate select, project, union, intersection, set difference, and natural join relational operations using simple example relations provided.	C	
23	List similarities and differences between object-oriented database concepts and features and those of relational databases.	C	*Suitable?*
24	Explain the relationship between functional dependencies and keys and give examples.	C	
25	Explain how having normal form relations reduces or eliminates attribute redundancy and update/delete anomalies.	E	
26	Normalize a set of relations to at least fourth normal form	E	*FPM working group*
27	Explain the primary key requirements for relational integrity.	E	*FPM working group*
28	Define and explain the need for referential integrity.	E	
29	Give examples of user-defined integrity constraints.	E	
30	Describe and interpret Entity Relationship or UML data modeling diagrams.	D	
31	Create simple Entity Relationship or UML data modeling diagrams.	D	

32	Describe and interpret Enhanced Entity Relationship diagrams.			*Delete.*
33	Select appropriate business rules for a given scenario.	E		*Design?*
34	Design and defend your design for a relational database for a given scenario.	E		*FPM working group*
35	Describe the relationship between a logical model and a physical model.	D	E	
36	Select a database pattern or standard model that effectively corresponds to a given scenario.	D	E	
37	Explain the use of CASE tools in data modeling.	D		
38	Describe data integration.			*Delete.*
39	Describe meta-modeling.			*Delete.*
40	Describe a data warehouse, its basic structure, etc.			*Delete.*
41	Distinguish between data administration and database administration.	B		
42	Apply ethical principles to database design, development and use.	B		*FPM working group*
43	Explain the concept of database security.	B		
44	Explain the concept of backup and recovery.	B		
45	Distinguish between homogeneous, heterogeneous and federated distributed databases.			*Delete.*
46	Explain the concept of replication as it pertains to distributed databases.			*Delete.*
47	Distinguish between horizontal and vertical replication as it pertains to distributed databases.			*Delete.*
48	Describe a client-server database architecture.			*Delete.*
49	Describe an n-tier database architecture.	C	G	
50	Explain the role of ODBC, JDBC and XML in the implementation of an n-tier database architecture.	G		*Suitable?*
51	Describe the concept of web services and the role of SOAP.	G		*Suitable?*
52	Demonstrate an understanding of online analytical processing and data warehouse systems.			*Delete.*

Networks (Information Systems / Information Technology)
Enabling Learning Objectives
Prepared: October 22, 2009 (@ University of the Fraser Valley)

Original source for all learning outcomes are the:
"Network Centric" category of the ACM/IEEE 2001 CS Curriculum Proposal
"Networking" items from the ACM 2008 IS/IT Curriculum proposal

As each enabling learning objective corresponds to a single summary objective, the first occurrence of the summary objective will contain the entire text of that objective.

#	Enabling Learning Objective	Summary cross-reference	
1	Manage networked accounts	A	Manage a network for optimal performance, and troubleshoot the network.
2	Enhance network performance	A	
3	Protect servers from data loss	A	
4	Discuss the benefits of network management and planning	A	
5	Develop networking standards, policies, procedures and documentation	A	
6	Describe the main challenges faced in a modern office environment using networks	A	
7	Troubleshoot a network following a structured approach	A	
8	Discuss the types of specialized equipment and other resources available for troubleshooting	A	
9	Configure IPX access lists and SAP filters to control basic Novell traffic.	A	
10	Enable the Novell IPX protocol and configure interfaces.	A	
11	Monitor Novell IPX operation on the router.	A	
12	Apply basic data communication theory to the performance analysis of networks.	A	
13	Explain the OSI reference model	B	Know the major communication architectural models.
14	Explain the OSI reference model's layers and their relationships to networking hardware and software	B	

15	Discuss the layered architecture of protocols, and describe common protocols and their implementation	B	
16	Describe the different major network architectures, Compare and contrast them.	B	
17	Outline the limitations, advantages, and disadvantages of each standard or architecture	B	
18	Define network services	C	Identify specific architectural features in networks.
19	Discuss the differences between centralized and client/server computing	C	
20	Define the client/server networking environment	C	
21	Discuss the basics of Web-based computing environments	C	
22	Describe the basic concepts associated with wide area networks (WANs)	C	
23	Describe how to use the Internet for a private connection using VPNs	C	
24	Describe the benefits of virtual LANs.	C	
25	Discuss the criteria for Selecting the Right Type of Network	D	Construct networks ranging from small Lans to large WANs.
26	Discuss the criteria for Selecting a Topology	D	
27	Describe the basic steps required for network operating system installation	D	
28	Install and configure network applications	D	
29	Create a network security plan	D	
30	Describe WAN protocols, and software and hardware technologies to build WANs.	D	
31	Design, build, and maintain a small local area network	D	
32	Describe the process of setting up peer-to-peer to networks.	D	
33	List commands to configure Frame Relay LMIs, maps, and subinterfaces.	D	
34	List commands to monitor Frame Relay operation in the router.	D	
35	Describe the differences between Local and Wide Area Networks	E	Know the definitions for a broad range of network terms.

36	Provide definitions for basic networking terms: Clients, Peers, Servers, the Network Medium, Network Protocols, Network Software, Network Services	E	
37	Describe the basic Network Types: Peer-to-Peer, Server-Based, Storage-Area Networks (SANs), Personal Area Networks (PANs), Hybrid Networks, Server Hardware Requirements, Specialized Servers	E	
38	Define and understand technical terms related to cabling, including attenuation, crosstalk, shielding, and plenum	E	
39	Describe a range of network topologies	F	Know the different network devices, transmission media and topologies for combining them into networks.
40	Describe the basic types of Hubs: Active Hubs, Passive Hubs, Hybrid Hubs	F	
41	Identify three major types of network cabling and of wireless network technologies	F	
42	Decide what kinds of cabling and connectors are appropriate for particular network environments	F	
43	Explain how network adapters prepare data for transmission, accept incoming network traffic, and control how networked communications flow	F	
44	Explain how larger networks may be implemented using devices such as repeaters, bridges, routers, brouters, gateways, and switches	F	
45	Configure routers to setup different types of LANs and WANs using LAN and WAN protocols.	F	
46	Describe the advantages and methods of network segmentation.	F	
47	Name and describe two switching methods.	F	
48	Describe full- and half-duplex Ethernet operation.	F	
49	Describe the features and benefits of Fast Ethernet.	F	
50	Explain and describe the characteristics of various transmission media.	F	

51	Contrast base band and broadband transmission technologies	G	Know a wide variety of data communication protocols and know the advantages and disadvanteges of each one.
52	Describe wireless transmission techniques	G	
53	Describe rudimentary signaling technologies for mobile computing	G	
54	Explain the IEEE 802 networking model and related standards	G	
55	Describe the function and structure of packets in a network, and analyze them	G	
56	Explain the function of protocols in a network (e.g., TCP/IP	G	
57	Describe various channel access methods, Compare and contrast them.	G	
58	Discuss the different types of carriers used for long-haul network communications	G	
59	Identify virtual LANs, LAN switching, Fast Ethernets, Frame Relay, ISDN networking.	G	
60	Identify the uses, benefits, and drawbacks of advanced WAN technologies such as ATM, FOOI, SONET, and SMDS	G	
61	List the required IPX address and encapsulation type.	G	
62	Describe network congestion problem in Ethernet networks.	G	
63	Distinguish between cut-through and store-and-forward LAN switching.	G	
64	Describe the operation of the Spanning Tree Protocol and its benefits.	G	
65	Differentiate between the following WAN services: LAPB, Frame Relay, ISDN/LAPD, HDLC, PPP, and DDR.	G	
66	Recognize key Frame Relay terms and features.	G	
67	Identify PPP operations to encapsulate WAN data on Cisco routers.	G	
68	State a relevant use and context for ISDN networking.	G	
69	Identify ISDN protocols, function groups, reference points, and channels.	G	
70	Describe Cisco's implementation of ISDN BRI	G	
71	Explain and identify key protocol information given samples of captured packets.	G	

72	Describe today's data communications industry as a system of interconnected components.	**(Recommend removal)**	
73	Recount network adapter enhancements that can improve performance	**(Recommend removal)**	
74	Implement the LAN protocols. ???	**(Recommend removal)**	
75	Implement TCP/IP. ???	**(Recommend removal)**	
76	Understand the standards governing network architectures	**(Recommend removal)**	
77	Understand the various networking software components	**(Recommend removal)**	
78	Discuss interconnectivity issues in a multivendor environment	**(Recommend removal)**	
79	Define the various options to implement a multivendor network environment	**(Recommend removal)**	
80	Understand the various alternatives used in network communications	**(Recommend removal)**	
81	Explain the role of driver software in network adapters	**H**	Know how a network OS works and be able to install and configure it.
82	Explain the operation fundamentals of network operating systems	**H**	
83	Provide a basic overview of networks, at the highest level.	**I**	Provide a basic overview of networks, at the highest level.