Final Report

British Columbia Computing Education Committee

Flexible Pre-Major Implementation Project

2012 November 30

Revised 2013 June 4

Project Lead: Rick Gee

Table of Contents

Executive Summary

From December 2007 to December 2009 a subcommittee of the British Columbia Computing Education Committee (BCCEC) undertook a Flexible Pre-Major (FPM) analysis project, resulting in the conclusion, described in [Zastre], that an FPM in computing was possible, an FPM that focused on learning outcomes as a mechanism for comparing the lower-level of programs for equivalence.

More correctly, the report noted that two FPMs are possible, one for Computer Science[1] (for students transferring into the third year of a Computer Science major at a university) and one for Computer Information Systems[2] (for students transferring into the third year of an applied degree program at a university or a college.)

From January 2010 to October 2012, essentially the same subcommittee (augmented by other interested instructors and professors) identified outcomes which should be part of one or both of the FPMs. The major task was to revise and rationalize the learning outcomes (which had been developed as part of the analysis project), to determine which are applicable to which FPM, and to determine the level at which the students should meet the outcomes.

This was completed in the spring of 2012.

Once the outcomes were identified and organized, institutions identified courses they offered which met these outcomes. This was completed in the fall of 2012.

This was followed by obtaining institutional consensus and publicizing the benefits of the approach.

[1] Computer Science is the term used at UBC, UVic, and UNBC. Computing Science is the term used at SFU.
[2] Computer Information Systems is the term used for the applied degrees at University of the Fraser Valley and Okanagan College.

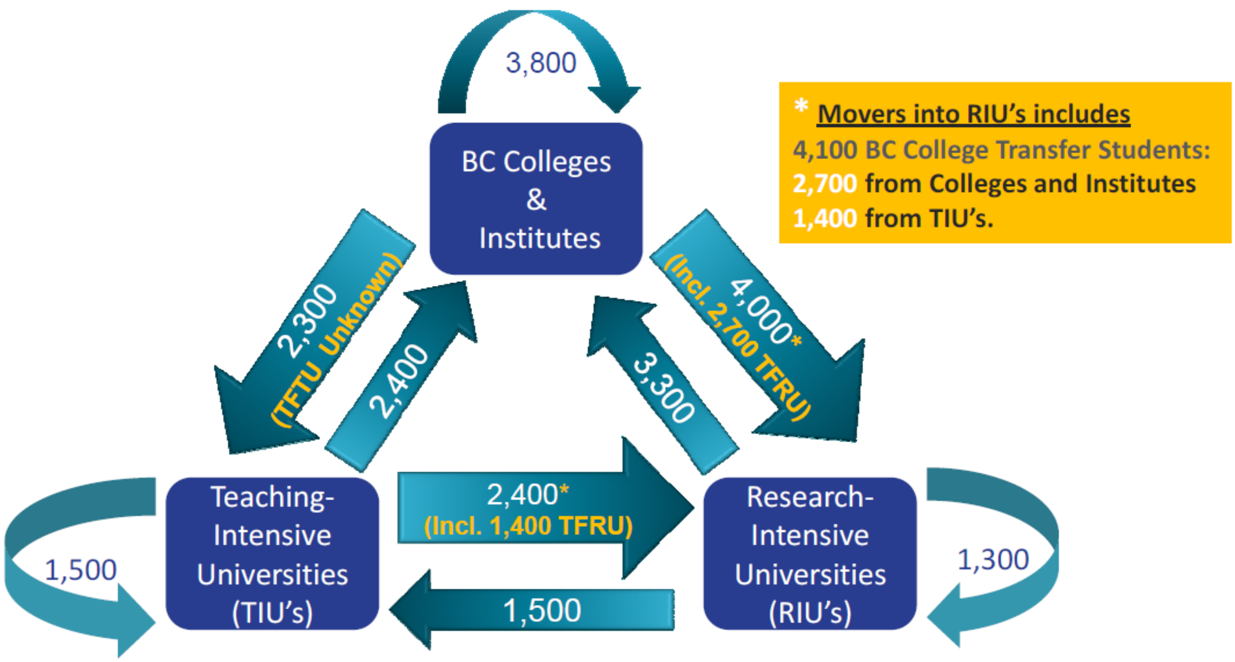
Background and Objectives

Background

As described in [Zastre], BCCEC consists of representatives from post-secondary institutions in British Columbia which offer computer science, computing science, and computer information systems courses and programs. The members include teaching-intensive universities, research-intensive universities, community colleges and institutes, and private post-secondary institutions.

There are many different ways to teach the first computing courses [CS2001] and members of BCCEC use many of them. Many students start at one institution and transfer to another (or to others) before completing their programs. Thus, students have difficulty transferring from one institution to another given that different pedagogical approaches can translate into significantly different courses.

While we would like to be able to provide detailed statistics on student transfers those detailed statistics are not available and we must rely on anecdotal information and general information. This was discussed in [Zastre, page 6]. Since that document was completed, [PSM] has provided some statistics.



The details behind the above diagram are available in [PSM].

Our BCCAT System Liaison Person, Neil Coburn, suggested a project to ease the transition between institutions. BCCAT approved BCCEC's proposal and the Analysis Project began in December 2007.

It was completed in May 2009. BCCEC subsequently applied for funding for an Implementation Project. While awaiting approval, which came on 19 October 2010, the project committee began work. This is the Final Report of their work.

Definition of an FPM

Since the reader may be unsure of the meaning of the term Flexible Pre-Major, we begin with a definition, from [FPM Working Group, page 6].

> "Pre-Major refers to the specific lower level pre-requisite courses to third year major courses. A Flexible Pre-Major is a set of flexible requirements that is, a) deliverable by sending institutions and acceptable to receiving institutions, and b) deemed to fulfill the lower level requirements for the major. The nucleus of the FPM is an agreement on a set of courses that all receiving institutions will accept in lieu of their own specific course requirements. The aim of the agreement is to sufficiently prepare students to enter a major program at the third year level with reasonable prospects of academic success. FPMs are generally expected to work in conjunction with Associate Degrees or other models where students transfer after completing 60 credits prior to transfer, although students may be able to transfer successfully into a major at receiving institutions with fewer credits. The FPM is a formal inter-institutional agreement facilitating student transfer into majors and is usually accompanied by a grid of equivalent courses for each category of the major or some similar description of the courses accepted as pre-major equivalents. The FPM does not guarantee acceptance into a program or major by the receiving institution since admission is related to other factors such as GPA. The FPM simply indicates that the student has covered off the lower level requirements for a major in a specific discipline as agreed upon by the articulation committee."

Note that an FPM in a subject area includes only those topics **within the subject area** that must be completed in the first two years. For example, the Computer Science FPM includes only the Computer Science topics students must complete; it does not include the Mathematics, English, or science courses many universities require nor does it address the breadth requirements of universities like SFU.

Where the quotation above refers to 60 credits, it is assuming that every course is worth three credits; 60 credits is therefore 20 courses, a normal full set of courses in the first two years of baccalaureate study.

At the time of writing, UVic uses units instead of credits. 1.5 units = 3 credits.

Project Objectives

Since [Zastre] indicated that an FPM was achievable for computer science and that an FPM was achievable for computer information systems, this Implementation Project had two major goals:

1. To develop an FPM for computer/computing science.
2. To develop an FPM for computer information systems.

Both of these FPMs include their adoption by the institutions involved and developing a process to ensure the FPMs are kept up to date.

From the contract between BCCEC and BCCAT for this Implementation Project, the complete list of deliverables for the project follows.

1. A summary of the student transfer patterns in Computer Science and Computer Information Systems
2. An updated version of the Learning Outcomes outlined in the Appendices to the FPM Analysis Report
3. A definition of the Flexible Pre-Majors in the Computer Science and Information System streams. As institutions are used to course-by-course transfer, this definition will list the learning outcomes for an FPM and will also include a list of courses (a "basket of courses") for each institution, the sum of whose outcomes matches [a defined percentage of] those of the FPM.
4. Grids for the two FPMs that outline the specific courses that make up the "basket" at each post-secondary institution
5. A rationale section that explains how the Flexible Pre-Majors will assist students, as well as post-secondary institutions
6. A list of BC post-secondary institutions that have agreed to implement one or both of the Flexible Pre-Majors, with evidence of formal agreement in the form of institutional signoffs
7. Description of a process for the BCCEC to review and update the two Flexible Pre-Majors on a regular basis

Project Team

The members of the project team which developed the Analysis Report were so excited about the project that all members wanted to continue to the implementation team. Unfortunately, institutional changes prevented some members from continuing. Those who continued were joined by other BCCEC members. Participating individuals (alphabetically by last name within institution) and their institutions (alphabetically) were:

Alexander College: Gordon Simon

British Columbia Institute of Technology (BCIT): Bill Klug, Brian Pidcock

Langara College: Mingwu Chen, Bryan Green

Okanagan College: Rick Gee

Selkirk College: Rita Williams

Simon Fraser University (SFU) Burnaby campus: Diana Cukierman, Anne Lavergne

Thompson Rivers University (TRU): Mohd Abdullah, Surinder Dhanjal

University of British Columbia (UBC) Vancouver campus: Donald Acton, Ed Knorr

University of the Fraser Valley (UFV): Paul Franklin

University of Northern British Columbia (UNBC): David Casperson

University of Victoria (UVic): Michael Zastre

Naming an institution above does not imply support of the institution; it simply states that one or more employees of that institution was/were involved and provided personal perspective.

Alexander College was a new member of the project team, representing private colleges within BC, a perspective which was previously lacking.

Additional assistance and advice was provided by Jennifer Orum, Fiona McQuarrie, and John FitzGibbon from BCCAT.

As [FPM Working Group] points out, the person signing off the FPM at an institution may vary from institution to institution. For the sake of this report, we will refer to that person as the registrar. Unfortunately the BC Registrars Association has spoken against including FPM notation on the transcript [FPM Working Group, page 8]. See also the appendices for the opinions of several registrars.

Thus, missing from the project team was a registrar. Our decision was to leave registrar consultation at the level of committee-member exchanges with their institution's Office of the Registrar.

The project lead was Rick Gee.

Problem Statement

Given the learning outcomes identified in [Zastre], how can they be improved and then combined to produce a usable guide?

Behind that simple question is a great deal of discussion and other work which is described in two of the outcomes in the Implementation Project proposal.

1. A definition of the Flexible Pre-Majors in the Computer Science and Information System streams, most likely described in terms of the learning outcomes that need to be covered in the "basket of courses" for each stream
2. Grids for the two FPMs that outline the specific courses that make up the "basket" at each post-secondary institution

Upon completion of those two outcomes, other aspects of the other outcomes came into play. These include efforts to obtain department approval at each institution, obtaining upper-level administration approval at each institution, handling disagreements or concerns, etc. These are more fully described in [FPM Working Group, section 3 and Appendices III and IV].

We should note that there are several other FPMs underway in BC [Fiona McQuarrie, personal communication 2012-03-12].

- The Economics FPM has recently been completed.
- The English FPM is well-recognized and most post-secondary institutions in BC are participants.
- The Psychology FPM is recognized by approximately half the post-secondary institutions in BC.
- The Sociology and Anthropology FPMs are garnering signoffs.

There was a Music FPM but it was cancelled in May 2011. Details on other FPMs and their progress, or lack thereof, are provided in [FPM Working Group, Appendix II].

Process

Plan of Meetings

The project was funded for a series of in-person meetings (some official, some unofficial), as we had found that they were much more productive than meetings at a distance. Some were held in conjunction with BCCEC articulation meetings; others were held independently.

The meeting dates were:

2010 May 6 – Okanagan College, Kelowna: in conjunction with the BCCEC spring meeting

2010 October 21 and 22 – College of the Rockies, Cranbrook: in conjunction with the BCCEC fall meeting

2011 May 4 and 5 – College of New Caledonia, Prince George: in conjunction with the BCCEC spring meeting

2011 October 20 and 21 – Douglas College, New Westminster; in conjunction with the BCCEC fall meeting

2011 December 9 – Langara College, Vancouver.

2012 February 15 – Okanagan College, Kelowna: between Bryan Green and Rick Gee

2012 May 3 – UBC, Vancouver: in conjunction with the BCCEC spring meeting.

2012 October 25 and 26 – NWCC, Prince Rupert: in conjunction with the BCCEC fall meeting

For the first three meetings, the project team met prior to the BCCEC meeting and then BCCEC devoted a portion of its meeting time to discussion and break-out sessions on the various subject areas.

The 2011 December meeting was limited to the project team, who convened for an intense day of discussions at Langara College.

The 2012 February meeting involved Bryan (visiting family in Kelowna during his Reading Break) and Rick (based in Kelowna). Our specialities overlap but are not congruent, so Bryan was able to clarify some of the areas in which Rick was confused.

The 2012 May meeting resulted in some conceptual breakthroughs, which led BCCEC to request, and be granted, a six-month extension to the project.

The 2012 October meeting reviewed this report and set the course for the future.

Sources and Resources

The main resource was [Zastre] which contained the initial drafts of learning outcomes. Over the course of the meetings these outcomes were supplemented, trimmed, massaged, and otherwise processed. The first decision made was to remove the word "understand" from the outcomes. In the committee's opinion, you can't measure understanding.

The objectives involving "understand" were rephrased to more clearly describe the outcome in a manner providing better guidance to those writing evaluation instruments and to those comparing academic programs.

Revising the outcomes

As project lead, Rick Gee took on the task of managing a significant revision of the learning outcomes. This included clarifying wording, adding and deleting outcomes, and ordering the outcomes. This was done via frequent consultation with other members of the project team.

Combining the outcomes

The outcomes were derived from eight subject areas.

- Algorithms and Data Structures
- Computer Architecture
- Hardware
- Information Management
- Introductory Programming
- Networking
- Software Engineering
- Web Learning

All of these areas are deemed appropriate for studies at the first- and second-year levels.

Note that there is some overlap between and among these areas. Notable examples include Computer Architecture/Hardware/Networking and Introductory Programming/Algorithms and Data Structures/Software Engineering.

The FPM committee identified the following areas for the FPM for Computer Science.

- Algorithms and Data Structures
- Computer Architecture
- Introductory Programming
- Software Engineering

The FPM committee identified the following areas for the FPM for Computer Information Systems.

- Algorithms and Data Structures

- Hardware
- Information Management
- Introductory Programming
- Networking
- Software Engineering
- Web Learning

The FPM for Computer Science includes fewer areas since those students have additional breadth requirements (English writing skills, Mathematics, Sciences, foreign languages, etc.) that are more extensive than those for students in Computer Information Systems.

Enabling outcomes and summary outcomes

Bloom's taxonomy in the cognitive domain [Bloom] has various levels, including (from simplest through most-complex): knowledge, comprehension, application, analysis, synthesis and evaluation. Knowledge outcomes are relatively simple, sometimes involving only memorization, while evaluation outcomes (the highest level) involve making complex judgments.

These categories of Bloom's taxonomy of the cognitive domain are from Bloom's original work. Since adopting those categories, the FPM committee has become aware, via [Johnson], of a 2001 revision, in [Anderson], to the names of the levels. These new names are remembering, understanding, applying, analysing, evaluating, and creating. The FPM has chosen to continue to use the original terms.

While providing examples of outcomes [Bloom] uses verbs applicable to many areas of study. These verbs are used in the enabling outcomes. For example, [BCIT] includes the verbs define, identify, label, list, name, recall, and state as suitable for outcomes at the knowledge level. Describe, discuss, explain, locate, paraphrase, give an example, and translate are suitable for outcomes at the comprehension level.

[Fuller at al] and [Gluga et al] provide examples specifically from Computer Science and Computer Information Systems.

For each of the eight areas above, we developed lists of *enabling outcomes*. These are low level outcomes that define quite precisely what it means to say the student has achieved the outcome. These are (usually) cognitive and (occasionally) psychomotor outcomes; we did not explore affective outcomes.

For each area, the list of enabling outcomes typically included 100 or more entries. But a list of over 100 outcomes is an intimidating list. For each area we have summarized the enabling outcomes into a smaller set of *summary outcomes*. In the spreadsheets underlying this report, each summary outcome lists its enabling outcomes. The enabling outcomes are not included in this report but are available electronically.

Our intent is that programs are to be compared at the level of summary outcomes. Where uncertainty or disagreement exists when using a summary outcome, departments will refer to the corresponding enabling outcomes.

Duplication of outcomes

While the subject areas are distinct, there have been a number of cases where the same or similar outcomes came from two areas. The most common overlap is in the Introductory Programming, Software Engineering, and Algorithms areas.

Duplicates have been identified.

A draft of the outcomes was included in [Zastre]. The final outcomes (organized alphabetically and by FPM) appear in the appendices to this report.

Not all outcomes have the same weight

It is not appropriate to say that a student has completed an FPM when she/he has completed X% of the outcomes. That X% may involve very many outcomes appearing lower in Bloom's taxonomy with relatively fewer higher in the taxonomy.

However, in our committee's opinion it is appropriate to say that a student has completed an FPM when she/he has completed X% of the knowledge and comprehension outcomes and Y% of the other outcomes. What should X and Y be?

Current transfer practices, confirmed at the October 2012 BCCEC meeting, imply that X and Y are in the range of 70-80% [Zastre, pp 15-6].

Based on intimate acquaintance with the outcomes, the author suggests that X should be 80% and Y should be 70%. That is, a student has completed an FPM if she/he has completed 80% of the knowledge and comprehension outcomes listed and 70% of the other outcomes listed. The October 2012 BCCEC meeting confirmed these percentages.

The summary outcomes are listed in the appendices, organized alphabetically and by FPM.

Translating outcomes into courses or vice versa

Student transcripts are written in terms of courses, not of learning outcomes. Thus it will be easier for students to transfer if they can provide either a statement that they have completed an FPM or they can prove they have completed a collection of courses which together satisfy the FPM outcomes.

Notation

The committee feels a notation on a transcript would be preferable; however, there is institutional resistance to the addition of notations on transcripts.

Until the May 2012 BCCEC meeting, the recommendation of this report was to be that departments provide letters to students stating they have completed the FPM at the sending institution and the department at the receiving institution will need to deal with their Registrar's Office to ensure the correct course transfer is applied.

By incorporating the baskets of courses (described more fully below) into an FPM recorded by BCCAT, the necessity for department letters has disappeared.

Baskets

Proving completion of a collection of courses is the more traditional approach and hence should be accommodated. This is one of the deliverables of this Implementation Project.

The following table identifies the courses which each institution has identified as providing the learning outcomes for the FPMs. In the second column of the table, the courses on one line are alternatives which both provide the same or sufficiently similar outcomes. Courses on separate lines each provide outcomes towards the FPM, and the courses on separate lines must all be completed successfully for the student to earn the FPM.

The Computer/Computing Science FPM does not specify a discrete mathematics course as the FPM focuses on Computer/Computing Science courses. For the convenience of all concerned, many institutions have specified discrete mathematics courses that students should take.

Note that certain institutions (Grande Prairie Regional College, Mount Royal University, Northern Alberta Institute of Technology, Red Deer College, and Southern Alberta Institute of Technology) are included in this table as they have participated in BCCEC in the past and as there is movement of students between BC and Alberta. These institutions are not members of the BC transfer system and will thus not be signing off on these FPMs.

Note that certain institutions which are members of the BC transfer system have not been included as they have not participated in BCCEC recently or do not offer courses towards the FPM.

Institutions whose names are in italics support the concept of the Computer Science Flexible Pre-Major but do not offer students all the courses to complete it at this time. Students may investigate institutions offering online courses to complete their missing courses.

FPM for Computer/Computing Science	
Institution	Courses
Alexander College	CPSC 111 Introduction to Computation CPSC 112 Introduction to Programming CPSC 115 Discrete Structures

Athabasca University	COMP 206 Introduction to Computer Programming (C++) or COMP 268 Introduction to Computer Programming (Java) COMP 272 Data Structures and Algorithms (Java) COMP 410 Software Engineering MATH 309 Discrete Mathematics
BCIT	COMP 1510 Programming Methods COMP 2721 Computer Organization/Architecture COMP 3760 Algorithm Analysis and Design COMP 8081 Management Issues in Software Engineering
Camosun College	COMP 132 Programming Using Java COMP 139 Applied Computer Programming COMP 182 Architecture and Programming COMP 210 Data Structures and Algorithms COMP 235 Software Engineering
Capilano University	COMP 121 Fundamentals of Programming COMP 126 Principles of Software Design COMP 134 Programming in Java COMP 210 Data Structures and Abstraction COMP 211 Computer Design and Architecture I COMP 212 Computer Design and Architecture II COMP 213 Introduction to Software Engineering MATH 124 Discrete Mathematics
College of New Caledonia	CSC 109 Computing Science I CSC 110 Computing Science II CSC 115 Discrete Computational Mathematics I CSC 212 Object-Oriented Software Development CSC 214 Introduction to Computer Systems CSC 215 Discrete Computational Mathematics II CSC 216 Introduction to Data Structures CSC 218 Introduction to Software Engineering CSC 224 Computer Organization
College of the Rockies	COMP 105 Introduction to Programming in the C and C++ Languages COMP 106 Intermediate C++, 3D Graphics, and Numerical Methods

Columbia College*	CSCI 120 Introduction to Computing Science and Programming I CSCI 125 Introduction to Computing Science and Programming II CSCI 150 Introduction to Digital and Computer System Design CSCI 225 Data Structures and Programming CSCI 250 Introduction to Computer Architecture CSCI 275 Software Engineering
Coquitlam College	CSCI 120 Introduction to Computer Science and Programming I (Python) CSCI 125 Introduction to Computer Science and Programming II (Java) CSCI 150 Introduction to Computer Design CSCI 201 Data & Program Organization or CSCI 225 Data Structures and Programming CSCI 275 Software Engineering MACM 101 Discrete Mathematics I
Douglas College	CMPT 1110 Introduction to Computing Science Using C++ CSIS 1275 Java Programming CSIS 2475 Data and Control Structures MATH 1130 Discrete Mathematics I MATH 2230 Discrete Mathematics II
Grande Prairie Regional College (AB)	CS 1140 Introduction to Computing Science CS 1150 Elementary Data Structures CS 2010 Practical Programming Methodology CS 2290 Computer Organization and Architecture I CS 3290 Computer Organization and Architecture II
Kwantlen Polytechnic University	CPSC 1103 Introduction to Computer Programming I CPSC 1204 Introduction to Computer Programming II CPSC 1250 Introduction to Computer Design CPSC 2302 Data Structures and Program Organization CPSC 2405 Introduction to Discrete Mathematics I
Langara College	CPSC 1150 Program Design CPSC 1160 Algorithms and Data Structures I CPSC 1181 Object-oriented Computing CPSC 2150 Algorithms and Data Structures II One of CPSC 2180 Computing Architecture or 2401 Digital Systems Design CPSC 2190 Theoretical Foundations of Computer Science

Mount Royal University (AB)	COMP 1501 Programming I: Introduction to Problem Solving and Programming COMP 1502 Programming II: Object Oriented Programming COMP 2503 Programming III: Data Structures COMP 2531 Computer Architecture and Operating Systems
North Island College	CPS 100 Computer Programming I CPS 101 Computer Programming II CPS 212 Discrete Mathematics & Computer Science
Northern Alberta Institute of Technology	CMPE 1300 Fundamentals of Programming CMPE 1600 Event-driven Programming CMPE 1700 Data Structures and Algorithms CMPE 2300 Object-Oriented Programming
Northern Lights College	CPSC 111 Computer Science and Information Technology CPSC 122 Introduction to Object Oriented Programming C++
Northwest Community College	CPSC 123 Computer Programming
Okanagan College	COSC 111 Computer Programming I COSC 121 Computer Programming II COSC 211 Machine Architecture One of COSC 221 Introduction to Discrete Structures or MATH 251 Introduction to Discrete Structures COSC 222 Computer Data Structures
Red Deer College (AB)	Relevant programs have been suspended until further notice.
Selkirk College	Selkirk College offers only two courses towards the CS FPM. CPSC 100 Introduction to Programming I CPSC 101 Introduction to Programming II
Simon Fraser University (Burnaby)	Both CMPT 120 Introduction to Computer Science and Programming I and CMPT 125 Introduction to Computing Science and Programming II, or CMPT 126 Introduction to Computing Science and Programming CMPT 150 Introduction to Computer Design CMPT 225 Data Structures and Programming CMPT 250 Introduction to Computer Architecture CMPT 275 Software Engineering I MACM 101 Discrete Mathematics I
Southern Alberta Institute of Technology	SAIT appears to have no courses which apply to this FPM.

Thompson Rivers University	COMP 1130 Computer Programming 1 COMP 1230 Computer Programming 2 COMP 1380 Discrete Structures 1 for Computing Science or MATH 1700 Discrete Mathematics 1 COMP 1390 Discrete Structures 2 for Computing Science or MATH 1390 Discrete Structures 2 for Computing Science COMP 2130 Introduction to Computer Systems COMP 2230 Data Structures, Algorithm Analysis and Program Design COMP 3520 Software Engineering
Trinity Western University	CMPT 140 Introduction to Programming CMPT 150 Introduction to Discrete Math CMPT 166 Intermediate Programming CMPT 231 Data Structures and Algorithms CMPT 242 Computing Machine Organization
UBC (Vancouver campus)	CPSC 110 Computation, Programs, and Programming CPSC 121 Models of Computation CPSC 210 Software Construction CPSC 213 Introduction to Computer Systems CPSC 221 Basic Algorithms and Data Structures
UBC (Okanagan campus)	COSC 111 Computer Programming I COSC 121 Computer Programming II COSC 211 Machine Architecture COSC 221 Introduction to Discrete Structures COSC 222 Data Structures
University of Northern BC	CPSC 100 Computer Programming I CPSC 101 Computer Programming II CPSC 141 Discrete Computational Mathematics CPSC 200 Algorithm Analysis and Development CPSC 222 Introduction to Concurrent and Distributed Programming CPSC 230 Introduction to Logic Design CPSC 231 Computer Organization and Architecture CPSC 242 Mathematical Topics for Computer Science CPSC 260 Ethics in Computing Science CPSC 281 Data Structures I

University of the Fraser Valley	COMP 125 Principles of Computing One of COMP 150 Introduction to Programming or COMP 152 Introduction to Structured Programming COMP 155 Object-oriented Programming COMP 251 Data Structures and Algorithms COMP 256 Introduction to Machine Architecture
University of Victoria	CSC 110 Fundamentals of Programming I CSC 115 Fundamentals of Programming II CSC 225 Algorithms and Data Structures CSC 230 Introduction to Computer Architecture MATH 122 Logic and Foundations SENG 265 Software Development Methods
Vancouver Island University	CSCI 160 Computing Science I CSCI 161 Computing Science II CSCI 260 Data Structures CSCI 261 Computer Architecture & Assembly Language CSCI 265 Software Engineering
Yukon College	CPSC 128 Object-Oriented Programming I CPSC 129 Object-Oriented Programming II MATH 130 Finite Mathematics

Institutions whose names are in italics support the concept of the Computer Information Systems Flexible Pre-Major but do not offer students all the courses to complete it at this time. Students may investigate institutions offering online courses to complete their missing courses.

FPM for Computer Information Systems	
Institution	Courses
Alexander College	CPSC 111 Introduction to Computation CPSC 112 Introduction to Programming CPSC 115 Discrete Structures
Athabasca University	COMP 206 Introduction to Computer Programming (C++) or COMP 268 Introduction to Computer Programming (Java) COMP 266 Introduction to Web Programming COMP 272 Data Structures and Algorithms (Java) COMP 347 Computer Networks COMP 361 Systems Analysis and Design COMP 378 Introduction to Database Management

BCIT	COMP 1510 Programming Methods COMP 1536 Introduction to Web Development COMP 2714 Relational Database Systems COMP 2721 Computer Organization/Architecture COMP 3721 Introduction to Data Communications COMP 3760 Algorithm Analysis and Design
Camosun College	COMP 132 Programming Using Java COMP 139 Applied Computer Programming COMP 155 Database Concepts COMP 162 Intro to Computers and the Web COMP 173 Computer Network Programming COMP 182 Architecture and Programming COMP 210 Data Structures and Algorithms COMP 230 Systems Analysis and Design COMP 235 Software Engineering
College of New Caledonia	CSC 109 Computing Science I CSC 110 Computing Science II CSC 115 Discrete Computational Mathematics I CSC 212 Object-Oriented Software Development CSC 214 Introduction to Computer Systems CSC 215 Discrete Computational Mathematics II CSC 216 Introduction to Data Structures CSC 218 Introduction to Software Engineering CSC 224 Computer Organization
College of the Rockies	COMP 105 Introduction to Programming in the C and C++ Languages COMP 106 Intermediate C++, 3D Graphics, and Numerical Methods COMP 155 Database Management COMP 165 Introduction to Web Programming
Columbia College	CSCI 120 Introduction to Computing Science and Programming I CSCI 125 Introduction to Computing Science and Programming II CSCI 150 Introduction to Digital and Computer System Design CSCI 225 Data Structures and Programming CSCI 250 Introduction to Computer Architecture CSCI 275 Software Engineering
Coquitlam College	Coquitlam College does not offer courses to meet the requirements of the CIS FPM

Douglas College	CMPT 1110 Introduction to Computing Science Using C++ CSIS 1150 Business Data Communications & Networking CSIS 1155 Hardware maintenance Concepts CSIS 1275 Java Programming CSIS 1280 Multimedia Web Development CSIS 2300 Database Management Systems
Grande Prairie Regional College (AB)	CS 1140 Introduction to Computing Science CS 1150 Elementary Data Structures CS 2000 Data Communications and Networking CS 2010 Practical Programming Methodology CS 2210 Introduction to PC Hardware and Systems Configuration CS 2910 Introduction to File and Database Management CS 3610 Systems Analysis and Design CS 3990 Topics in Internet Technologies
Kwantlen Polytechnic University	CPSC 2302 Data Structures and Program Organization INFO 1111 Introduction to Computer Hardware and Software INFO 1112 Principles of Program Structure and Design INFO 1113 System Analysis and Design INFO 1212 Networking Technologies I INFO 1213 Web Application Development INFO 1214 Discrete Mathematics for Information Technology INFO 2311 Networking Technologies II INFO 2312 Database Management Systems INFO 2313 Object Oriented Programming
Langara College	CPSC 1030 Web Development I CPSC 1150 Program Design CPSC 1160 Algorithms and Data Structures I CPSC 1280 Unix Tools and Scripting CPSC 1480 Networking One of CPSC 2030 Web Development II or CPSC 2261 Web Technology CPSC 2221 Data Base Systems CPSC 2301 Software Engineering CSIS 1410 Fundamentals of Microcomputers

North Island College	CPS 100 Computer Programming I CPS 101 Computer Programming II CPS 120 Introduction to PC Communications CPS 146 Database Fundamentals CPS 151 Systems Analysis & Design CPS 165 Web Design Tools CPS 180 PC Hardware & Troubleshooting CPS 212 Discrete Mathematics & Computer Science CPS 236 Internet Programming CPS 262 Data Communications & Computer Networks
Northern Alberta Institute of Technology	CMPE 1300 Fundamentals of Programming CMPE 1600 Event-driven Programming CMPE 1700 Data Structures and Algorithms CMPE 2300 Object-Oriented Programming
Northern Lights College	CPSC 111 Computer Science and Information Technology CPSC 122 Introduction to Object Oriented Programming C++
Northwest Community College	CPSC 123 Computer Programming
Okanagan College	COSC 111 Computer Programming I One of COSC 118 Networks and Telecommunications I or NTEN 117 Networks and Telecommunications I COSC 121 Computer Programming I COSC 126 Systems Analysis and Design One of COSC 150 Digital Logic and Microcomputer Hardware or NTEN 126 Digital Logic and Microcomputer Hardware COSC 211 Machine Architecture COSC 219 Client-side Web Systems COSC 222 Computer Data Structures COSC 304 Introduction to Database Management Systems
Red Deer College (AB)	Relevant programs have been suspended until further notice.
Selkirk College	CPSC 100 Introduction to Programming I CPSC 101 Introduction to Programming II
Southern Alberta Institute of Technology	SAIT appears to have no courses which apply to this FPM.

University of the Fraser Valley	CIS 145 Web Publishing CIS 190 Systems Hardware Concepts CIS 192 Introduction to Networking CIS 230 Databases and Database Management Systems CIS 270 Analysis and Design CIS 291 Networking Theory and Applications COMP 125 Principles of Computing One of COMP 150 Introduction to Programming or COMP 152 Introduction to Structured Programming COMP 155 Object-oriented Programming COMP 251 Data Structures and Algorithms COMP 256 Introduction to Machine Architecture
Yukon College	ICT 102 Computer Hardware ICT 106 Introduction to Programming ICT 108 Operating Systems I ICT 112 Foundations – Web Development ICT 114 Networking ICT 118 Operating Systems II ICT 214 Database Design ICT 216 Database Management

Following discussion at the spring 2012 BCCEC meeting, the decision was made to use the enabling and summary learning outcomes to identify baskets of courses and then incorporate those baskets into the FPMs. That is, the outcomes will be maintained by BCCEC as a guide to assist institutions in identifying the courses in their baskets. More and more institutions are formally stating the outcomes for their courses, so the translation of outcomes into courses should not be a problem. In fact, this translation may encourage all institutions to provide learning outcomes for their courses.

As a result, the FPM agreements for Computer/Computing Science and Computer Information Systems will look similar to those from other disciplines, in spite of the different path we have followed in identifying the courses.

In particular, we envisage the FPMs as consisting of a table containing two columns, as shown above.

BCCAT approval, Institutional signoff and beyond

BCCAT approval

Completion of this report does not mean the FPM is in place.

BCCAT must approve the FPM report. It is hoped this will happen at the first meeting of its Transfer and Articulation Committee (TAC) following the submission of this report. TAC is expected to meet in late January.

After receiving BCCAT's approval, the baskets of courses will be publicized through BCCAT's website.

While awaiting, and after receiving, BCCAT approval, institutional support will be obtained, using the signoff process described below.

Institutional signoff

[FPM Working Group, page 8] comments on institutional signoff.

> "… [T]he method for gaining agreement from an institution to participate in a FPM may vary by institution. For example, in some institutions, the Senate delegates the authority for these kinds of items to a standing committee or subcommittee. Since FPMs are not programs, the approval process might not be tied to a specific decision-making process in the institution. However, because the FPM is a formal inter-institutional agreement, it might go to the institutional governing body for approval, a process that would also enhance visibility of the agreement. Working Group members noted that a formal process would highlight agreements with senior staff in institutions but would also add significantly to the amount of time required to get all institutions to sign the agreement. The information gathered from the institutional decision-making maps and housed at BCCAT acts as a check for the signature gathering phase, i.e., collecting formal approval from each of the participating institutions. The Working Group developed a signoff sheet to be used by institutions to confirm their participation in specific FPMs."

The signoff sheet referred to in the last sentence is available in [FPM Working Group, Appendix VI].

Members of the project group will be approaching their institutions for adoption of this report. This report will be updated as adoptions happen.

Questions about signoff

If an institution offers all the courses necessary to meet the outcomes of the FPM, that institution will be able to consider signing off.

If an institution does not offer all the courses necessary to meet the outcomes of the FPM, can it sign off on the FPM? Yes, as long as it can direct the students to courses from other institutions which provide the missing outcomes [Fiona McQuarrie, personal communication 2012-05-02]. These other institutions may be neighbouring institutions, an option available where there are two or more local post-secondary institutions, or they may be institutions which offer distance courses.

Communications

Once institutional signoff is complete, the information about the FPM must be made available to the students. [FPM Working Group, pages 9 and 10] addresses this under the headings Student Advising and Communication about Flexible Pre-Majors.

> 7. Student Advising
>
> The goal of the FPM is to provide more information to students contemplating transfer directly into a major at another institution. In order to be useful, student information about FPMs needs to be clear, easy to use, reliable, and consistent. Therefore, the advising of students, both at their initial institution and at the one to which they are seeking to transfer is an important consideration. Providing information about the FPM is made more difficult when receiving institutions do not know students' intentions regarding entering a major until they actually declare it. It is difficult to track declaration of majors once a student is registered, as the admission to the major is often a departmental decision.
>
> Institutions that participate in a FPM may choose to advertise the existence of the agreement in a number of ways. For example, the institution may package the FPM with an Associate Degree, thereby encouraging students to complete and transfer with 60 credits and a credential. However, students who have already determined that they will transfer to another institution as soon as they have completed lower-level general degree requirements may choose to transfer before the full two years or 60 credits is completed. In institutions where students routinely transfer with less than 60 credits, the sending institutions might advise students how to complete FPM requirements before they transfer.
>
> In addition to institutional advising for students, FPMs will be included in the BC Transfer Guide as tags on courses that are listed as part of a FPM. When using the BC Transfer Guide search mechanism, if a student clicks on a 100 or 200 level course in a subject with a FPM agreement, for an institution that is participating, the search results will include an information box on FPMs and more information about the FPM in that discipline. The student can click the box and find out more about what a FPM is and the course grid for the discipline. A FPM will be included in the BC Transfer Guide when a critical mass of institutions has signed off. Only participating institutions will be referred to with other institutions added as they sign off on the agreement. Non-participating institutions will not be included.
>
> As articulation committees update the grid of courses on an annual basis, the information should be forwarded to BCCAT staff for inclusion in the BC Transfer Guide as well as any changes to the agreement resulting from the regular review.

8. Communication about Flexible Pre-Majors

Articulation committee faculty members have difficulties in drawing issues like FPM to the attention of senior staff in their institutions, thereby holding up the discussion and signoff processes for FPMs. The Working Group suggested that BCCAT act as the conduit for information on FPMs in process to the associations of registrars, academic vice-presidents, deans, governing body chairs, and others likely to be involved as decision-makers. This can be done through presentations or newsletters addressed to these associations, or through presentations to individual institutions or groups of institutions.

Students are the key group to communicate with regarding FPMs. This can be done by notes on the institutional calendar or website, on the BC Transfer Guide site, and in the advising offices of the institutions. Others that should know about FPMs are the deans of the disciplines involved, mostly Deans of Arts and Science. Academic VPs should be aware of the implications of FPMs and know that they are being implemented in their institutions. Generally, the responsibility for disseminating information regarding FPMs in any discipline lies with the articulation committee representative and the faculty members involved. In order to aid the communication process generally, the Group elaborated on a set of Frequently Asked Questions initially developed as part of the Psychology FPM.

As noted above, until the spring 2012 BCCEC meeting, the recommendation of this report was to be that departments provide letters to students stating they have completed the FPM at the sending institution and the department at the receiving institution will need to deal with their Registrar's Office to ensure the correct course transfer is applied.

By incorporating the baskets of courses into a FPM recorded by BCCAT, the necessity for department letters has disappeared. But the need for institutional approval remains.

Forms

[FPM Working Group, Appendix V, pages 21 and 22] provide a template you may use to determine the process within your institution for obtaining that approval.

[FPM Working Group, Appendix VI, pages 23 and 24] provide a template for an institutional signoff on the FPM. Based on that template, signoffs specific to the Computer Science (Appendix 8) and Computer Information Systems (Appendix 9) FPMs are provided.

Ongoing evaluation

The computing field continues to change at a rapid pace. For an FPM to maintain currency, some body or group needs to ensure that the FPM changes with the field. The obvious body or group is the BCCEC itself.

The recommendation is that the BCCEC meeting in the spring (late-April or early-May) should have a standing agenda item to review the Computer Science and Computer Information Systems FPMs. Whenever changes are deemed necessary, BCCEC will strike an ad hoc committee to make those changes and communicate them to all members of BCCEC.

Note that many computing (used in the broadest sense) principles remain the same but the implementation of them may change.

Principles change too and an FPM must be a living project.

Recommendations

That BCCEC adopt the Computer Science and Computer Information Systems FPMs as described in this report and continue to review them for currency.

BCCEC meetings in the spring (late-April or early-May) should have a standing agenda item to review the Computer Science and Computer Information Systems FPMs. Whenever changes are deemed necessary, BCCEC will strike an *ad hoc* committee to make those changes and communicate them to all members of BCCEC.

That BCCEC encourage post-secondary institutions and their departments to participate in the Computer Science and Computer Information Systems FPMs.

BCCEC members are encouraged to work with their registrars or other approvers to publicize and gain acceptance for the concept of FPM. This recommendation is in agreement with Recommendation 8 in [FPM Working Group].

That BCCEC member institutions, departments, and department members inform their students of the existence of the Computer Science and Computer Information Systems FPMs and encourage their students to complete the Computer Science and Computer Information Systems FPMs.

This recommendation is in line with Recommendation 9 in [FPM Working Group].

Acknowledgements

The project committee would like to thank:

- From the Analysis project
    - Michael Zastre (University of Victoria), for so ably leading the Analysis Project which provided us with a solid foundation on which to build
    - all other committee members
- From BCCAT
    - Jennifer Orum, for providing advice and guidance all through the process, until her retirement
    - Fiona McQuarrie, for providing advice and guidance following Jennifer's retirement
    - John FitzGibbon, for providing advice and guidance as the project progressed.

The editor would like to thank:

- All the members of the Implementation Project committee for their ideas, assistance, and time, especially
    - Bryan Green (Langara College) for acting as cheerful host of the project committee, for providing a crucial idea at just the right time, and for his editing of earlier versions of this document;
    - Ed Knorr (UBC Vancouver campus), for his editing of the spreadsheets and earlier versions of this document
    - Donald Acton (UBC Vancouver campus), for finding the ‘‘Can You Touch Your Toes” blog entry
    - Michael Zastre (University of Victoria), for his editing of earlier versions of this document.
- From BCCAT
    - Fiona McQuarrie, for providing advice and guidance following Jennifer's retirement
    - John FitzGibbon, for providing advice and guidance as the project progressed.

Thank you all.

References

[Anderson] "A taxonomy for learning, teaching and assessing: A revision of Bloom's Taxonomy of educational objectives: Complete edition". Anderson, L. W., & Krathwohl, D. R. (Eds.). Longman (New York) 2001.

[BCIT] "Writing Learning Outcomes". Anonymous. Undated. Available (as of 2012 11 05) at https://helpdesk.bcit.ca/fsr/teach/teaching/ja_learningoutcomes.pdf

[Bloom] "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 Electronics Engineers (IEEE) Computer Society, and the Association for Computing Machinery (ACM). December 2001. Available (as of 2012 11 05) at http://www.acm.org/education/curric_vols/cc2001.pdf

[FPM Working Group] "Flexible Pre-Majors: Final Report of the Flexible Pre-Majors Working Group". John FitzGibbon and Jennifer Orum in consultation with the Flexible Pre-Majors Working Group. BCCAT, 2011. Available (as of 2012 11 05) at http://www.bccat.ca/pubs/FPMFinalReport.pdf

[Fuller et al] "Developing a Computer Science-specific Learning Taxonomy". Ursula Fuller, Colin G. Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L. Lewis, Donna McGee Thompson, Charles Riedesel, Errol Thompson. Available (as of 2012 11 05) at https://www.cs.kent.ac.uk/pubs/2007/2798/content.pdf

[Gluga et al] "Coming to terms with Bloom: an online tutorial for teachers of programming fundamentals". Richard Gluga, Judy Kay, Raymond Lister, Sabina Kleitman, Tim Lever. This paper appeared at the Fourteenth Australasian Computing Education Conference (ACE 2012), Melbourne, Australia, January-February 2012. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 123. M. de Raadt and A. Carbone, Eds. Available (as of 2012 11 05) at http://crpit.com/confpapers/CRPITV123Gluga.pdf

[Johnson] "Multi-perspective survey of the relevance of the revised Bloom's taxonomy to an introduction to Linux course". Gregory Johnson, William Armitage, Alessio Gaspar, Naomi Boyer, Cliff Bennett. This paper appeared at the thirteenth SIGITE annual conference (SIGITE 2012), Calgary, AB. Available (as of 2012 11 05) at http://sigite2012.sigite.org/?presentation=multi-perspective-survey-of-the-relevance-of-the-revised-bloom%E2%80%99s-taxonomy-to-an-introduction-to-linux-course

[PSM] "Movers and Transfers in the BC Public Post-Secondary System". Post-Secondary Student Mobility (PSM) Subcommittee of STP . Available (as of 2012 11 05) at http://www.aved.gov.bc.ca/student_transitions/documents/PSM-Newsletter-2011.pdf

[Zastre] "Final Report: British Columbia Computing Education Committee Flexible Pre-Major Analysis Report". Michael Zastre. December 2009. Available (as of 2012 11 05) at http://www.bccat.ca/pubs/ComputingEducationFPMAnalysis.pdf

Appendices

Each outcome has a code identifying the subject area from which it is derived. These codes are:

- AL – Algorithms and Data Structures
- CA – Computer Architecture
- HW – Hardware
- IP – Introductory Programming
- IM – Information Management
- NW – Networking
- SE – Software Engineering
- WL – Web Learning

For summary outcomes, summarizing several enabling outcomes, the code is followed by a letter. The letter is a sequence letter, with no importance attached to the order in which that letter was assigned.

The enabling outcomes are identified by the same code, followed by a number.

In the summary outcomes that follow, the outcomes are listed alphabetically. Note that the verb "use" is not the verb "use" often identified as a suitable verb for Bloom's Application level. It is used here as a generic verb, to refer to a spectrum of activities coming from a variety of Bloom's levels. "Use" incorporates "identify", "create", "implement", "analyze", and "compare and contrast". Using it in this way has allowed us to simplify the summary outcomes.

Appendix 1 - Knowledge and Comprehension Summary Outcomes for Computer Science FPM

ALB	Define common terms used in Algorithms and Data Structures	AL9	AL10	AL23	AL32	AL33	AL34
		AL49	AL50	AL52	AL76	AL100	AL114
		AL121	AL129	AL130	AL155	AL156	
CAA	Define common terms used in Computer Architecture	CA1	CA2	CA17	CA18	CA36	CA37
		CA40	CA47	CA50	CA60	CA61	CA63
		CA64	CA70	CA72	CA75		
IPA	Define common terms used in Introductory Programming	IP14	IP20	IP21	IP25	IP26	IP77
		IP117	IP118	IP127			
SEB	Define common terms used in Software Engineering	SE1	SE2	SE5	SE6	SE13	SE14
		SE17	SE36	SE37	SE53		
CAE	Describe processors - single, multiple, parallel, specialized	CA5	CA26	CA27	CA28	CA29	CA38
		CA41	CA56	CA58	CA59	CA62	CA65
		CA66	CA67	CA73	CA74		
CAB	Describe the history of computer architecture	CA3	CA6	CA7			
SEM	Discuss issues arising in software deployment, maintenance and support.	SE44	SE50	SE51	SE52	SE54	
ALT	Discuss memory allocation issues	AL39	AL40	AL41	AL42	AL43	
CAF	Explain a system, at a holistic level	CA4	CA25	CA38	CA53	CA68	CA71

		CA76	CA82	CA83	CA84	CA85	
CAC	Explain basic electronics	CA8	CA9	CA10			
CAD	Explain data representation	CA11	CA12	CA13	CA14	CA15	CA16
		CA19	CA20	CA21	CA22	CA23	CA24
CAH	Explain interrupts	CA38	CA49				
SEI	Explain software lifecycles and their phases	SE44	SE45	SE46	SE47	SE48	SE49
		SE72					

Appendix 2 - Other Summary Outcomes for Computer Science FPM

Note that the verb "use" is being used in a generic sense and covers most levels of Bloom's taxonomy, rather than having many more summary outcomes, one for each level and for each data structure, algorithm, or technique.

ALD	Characterize code fragments/algorithms using time and space complexity, or as being recursive/iterative	AL12	AL13	AL14	AL20	AL21	AL22
		AL24	AL25	AL26	AL27	AL31	AL51
		AL60	AL80	AL82	AL85	AL88	AL90
		AL91	AL92	AL93	AL94	AL95	AL96
		AL97	AL98	AL106	AL112	AL113	AL121
		AL122	AL134	AL138	AL144	AL146	AL150
		AL151					
SEA	Complete a team-based project						
SEK	Construct high-quality software to realize a design	SE3	SE4	SE8	SE9	SE10	SE28
		SE38	SE39	SE49	SE59	SE64	SE72
SEG	Create and specify the software design for a medium-sized software project	SE11	SE12	SE13	SE15	SE16	SE18
		SE19	SE20	SE21	SE22	SE23	SE24
		SE25	SE26	SE27	SE28	SE29	SE30
		SE31	SE47	SE72			

ALA	Demonstrate mathematical literacy in the concepts applicable to Algorithms and Data Structures	AL1	AL2	AL3	AL4	AL5	AL6
		AL7	AL8	AL11	AL15	AL16	AL17
		AL18	AL19	AL53	AL54	AL114	AL123
		AL124	AL125	AL126	AL127	AL128	AL131
		AL132	AL133				
SEE	Demonstrate the central elements of team building and team management. (Software Engineering Management)	SE26	SE35	SE40	SE41	SE42	SE43
		SE64					
SED	Display competence with enabling technologies for software engineering	SE4	SE7	SE8	SE9	SE32	SE33
		SE34	SE38	SE39	SE46	SE75	SE76
SEH	Evaluate different designs prepared as solutions to the same problem.	SE11	SE13	SE15	SE16	SE18	SE19
		SE21	SE22	SE23	SE24	SE25	SE26
		SE27	SE28	SE29	SE30	SE31	SE47
		SE72					
ALS	Given a problem, identify an appropriate algorithm and/or data structure to solve it	AL47	AL48	AL69	AL70	AL71	AL72
		AL99	AL111	AL118	AL147	AL148	AL149
IPS	Perform complexity analysis	IP25	IP78	IP79	IP80	IP81	IP82
		IP83	IP84	IP85	IP86	IP92	
IPE	Perform simple input and output	IP10					
SEF	Produce a set of software requirements for a medium-sized software system (Requirements).	SE55	SE56	SE57	SE58	SE59	SE60

		SE61	SE62	SE63			
IPH	Read and explain code	IP1	IP2	IP3	IP4	IP5	IP28
		IP128					
SEC	Select, with justification, an appropriate set of tools to support the development of a particular software product. (Tools and Environments)	SE4	SE7	SE8	SE9	SE32	SE33
		SE34	SE38	SE39	SE46	SE75	SE76
		SE36	SE37				
SEJ	Select, with justification, the software development models and process elements for the development and maintenance of a particular software product	SE43	SE44	SE45	SE46	SE47	SE48
		SE49	SE72				
IPJ	Successfully deal with errors	IP28	IP29	IP30	IP31	IP32	IP65
		IP67	IP71	IP73	IP74		
SEL	Test code with unit tests, system tests, and user tests	SE61	SE65	SE66	SE67	SE68	SE69
		SE71	SE72	SE73	SE74	SE75	SE76
ALI	Use (binary) trees	AL61	AL66	AL67	AL68	AL79	AL101
		AL102	AL103	AL104	AL105	AL110	AL111
		AL115	AL116	AL117	AL119	AL120	
CAI	Use a bus	CA51	CA52	CA53	CA54		
IPM	Use collections	IP39	IP40	IP41	IP42	IP43	IP44
		IP45	IP46	IP47	IP87	IP88	IP89
		IP91	IP110				
IPD	Use conditional structures	IP4	IP10	IP11	IP12	IP13	IP125
		IP126					
IPV	Use dynamic programming	IP93					

ALE	Use finite state machines and regular expressions	AL152	AL153	AL154	AL157	AL158	AL159
		AL160					
IPN	Use generics/templates	IP51	IP52	IP53			
ALO	Use graphs	AL134	AL135	AL136	AL137	AL138	AL139
		AL140	AL141	AL142	AL143	AL144	AL145
		AL146					
ALK	Use hashing	AL59	AL60	AL62	AL66	AL67	AL68
		AL71	AL72	AL73	AL74	AL75	
ALN	Use heaps	AL63	AL65	AL66	AL67	AL68	
IPC	Use iteration	IP5	IP10	IP13	IP16	IP17	
IPK	Use language reference materials	IP33	IP34	IP35			
ALF	Use lists	AL35	AL44	AL55	AL56	AL57	
CAG	Use memory	CA39	CA38	CA41	CA42	CA43	CA44
		CA45	CA46	CA48	CA53	CA54	CA71
IPG	Use modelling tools and techniques (problem solving)	IP15	IP18	IP19	IP66	IP111	IP112
		IP113	IP114	IP115	IP27	IP90	
ALQ	Use O(n log n) sorts	AL90	AL91	AL92	AL93	AL94	AL95
		AL96	AL97				
ALP	Use O(n squared) sorts	AL81	AL82	AL83	AL84	AL85	AL86
		AL87	AL88	AL89	AL96	AL97	
IPU	Use object-oriented programming	IP103	IP104	IP105	IP106	IP107	IP108
ALJ	Use ordered trees, e.g., binary search trees, B trees, B+ trees	AL107	AL108	AL109	AL110	AL111	
CAJ	Use peripherals	CA57	CA54	CA49			
IPF	Use pointers and references	IP49	IP50				
ALM	Use priority queues	AL64	AL65	AL66	AL67	AL68	

IPI	Use procedures, functions, methods, subroutines a.k.a. Top-down design	IP10	IP22	IP23	IP24		
ALH	Use queues and deques	AL37	AL38	AL46	AL55	AL56	
ALC	Use recursion	AL77	AL78	AL79	AL80	AL129	
IPP	Use recursion	IP58	IP59	IP60	IP61	IP67	IP68
		IP69	IP70	IP119			
ALR	Use searching algorithms	AL28	AL29	AL30	AL57	AL58	AL59
IPL	Use simple datatypes (primitives)	IP36	IP37	IP38	IP124	IP125	
ALG	Use stacks	AL36	AL45	AL55	AL56		
IPO	Use testing, preconditions, postconditions, assertions	IP54	IP55	IP56	IP57	IP62	IP63
		IP64	IP65	IP126			
IPB	Write and test good code in more than one programming language	IP3	IP6	IP7	IP8	IP9	IP10
		IP48	IP72	IP122	IP123	IP121	IP129
		IP116	IP120	IP121	IP130	IP90	
IPQ	Write event-driven programs	IP72	IP73				
IPR	Write multi-threaded programs	IP74	IP75				
CAK	Write programs using assembly language	CA29	CA30	CA31	CA32	CA33	CA34
		CA35					

Summary outcome SEA (Complete a team-based project) has no specific enabling outcomes listed. Nonetheless, a student completing this FPM must have completed a team-based project.

Please note that there is some duplication in this table. In particular, the summary outcome "Use finite state machines" is repeated twice, as is "Use recursion". Ed Knorr has suggested more expansive wordings for these duplications:

“Use finite state machines in writing a program involving states and transitions” for Introductory Programming and “Use finite state machines in the design of an algorithm involving states and transitions” for Algorithms and Data Structures.

“Use recursion in the design of a non-trivial algorithm” for Algorithms and Data Structures and “Use recursion in the implementation of an algorithm” for Introductory Programming.

With these clarifications it is clearer that the levels at which students satisfy these outcomes will be lower in Introductory Programming than in Algorithms and Data Structures.

Appendix 3 - Knowledge and Comprehension Summary Outcomes for Computer Information Systems FPM

ALB	Define common terms used in Algorithms and Data Structures	AL9	AL10	AL23	AL32	AL33	AL34
		AL49	AL50	AL52	AL76	AL100	AL114
		AL121	AL129	AL130	AL155	AL156	
HWA	Define common terms used in hardware	HW11	HW24				
IMH	Define common terms used in Information Management	IM1	IM2	IM3	IM4	IM5	IM7
		IM12	IM13	IM14	IM15	IM17	IM18
		IM19	IM43	IM44	IM50	IM55	IM62
IPA	Define common terms used in Introductory Programming	IP14	IP20	IP21	IP25	IP26	IP77
		IP117	IP118	IP127			
NWI	Define common terms used in Networking	NW2	NW21	NW23	NW32	NW33	NW45
		NW46	NW47	NW48	NW49	NW55	

SEB	Define common terms used in Software Engineering	SE1	SE2	SE5	SE6	SE13	SE14
		SE17	SE36	SE37	SE53		
WLM	Define common terms used in Web Learning	WL4	WL5	WL9	WL10	WL13	WL14
		WL15	WL16	WL17	WL18	WL19	WL20
		WL35	WL36	WL48	WL52	WL54	WL59
HWI	Describe a motherboard	HW49	HW50	HW51	HW52		
IMB	Describe different database models and differentiate between them.	IM54					
IMA	Describe organizational needs relating to data acquisition, use, retention and disposition.	IM6	IM7	IM8	IM9	IM10	IM62
		IM63	IM64				
HWD	Describe primary memory	HW8	HW9	HW10	HW39	HW40	
HWC	Describe secondary storage	HW12	HW20	HW21	HW22	HW23	HW25
		HW26	HW27	HW28	HW29	HW30	HW37
		HW38	HW53	HW60			

HWE	Describe the CPU	HW6	HW7	HW43	HW45	HW46	HW47
		HW48					
NWB	Describe the major communication architectural models.	NW16	NW17	NW18	NW19	NW20	
HWB	Describe video	HW31	HW32	HW33	HW34	HW35	HW36
SEM	Discuss issues arising in software deployment, maintenance and support.	SE44	SE50	SE51	SE52	SE54	
ALT	Discuss memory allocation issues	AL39	AL40	AL41	AL42	AL43	
NWG	Explain how a network OS works and be able to install and configure one.	NW79	NW80				
SEI	Explain software lifecycles and their phases	SE44	SE45	SE46	SE47	SE48	SE49
		SE72					
NWC	Identify specific architectural features in networks.	NW22	NW24	NW25	NW26	NW27	NW28
		NW29	NW30	NW31			

NWF	List a wide variety of data communication protocols and describe the advantages and disadvantages of each.	NW63	NW64	NW65	NW66	NW67	NW68
		NW69	NW70	NW71	NW72	NW73	NW74
		NW75	NW76	NW77	NW78		
IMG	Use relational database terminology correctly	IM12	IM13	IM14	IM15	IM16	IM17
		IM18	IM19	IM21	IM23	IM25	IM27
		IM40	IM41	IM42	IM43	IM44	IM50
		IM52					

Appendix 4 - Other Summary Outcomes for Computer Information Systems FPM

Note that the verb "use" is being used in a generic sense and covers most levels of Bloom's taxonomy, rather than having many more summary outcomes, one for each level and for each data structure, algorithm, or technique.

WLC	Apply copyright law, ethics and internet law to a website.	WL12	WL63	WL75			
ALD	Characterize code fragments/algorithms using time and space complexity, or as being recursive/iterative	AL12	AL13	AL14	AL20	AL21	AL22
		AL24	AL25	AL26	AL27	AL31	AL51
		AL60	AL80	AL82	AL85	AL88	AL90
		AL91	AL92	AL93	AL94	AL95	AL96
		AL97	AL98	AL106			
SEA	Complete a team-based project						
SEK	Construct high-quality software to realize a design	SE3	SE4	SE8	SE9	SE10	SE28
		SE38	SE39	SE49	SE59	SE64	SE72
NWD	Construct networks ranging from small LANs to large WANs.	NW34	NW35	NW36	NW37	NW38	NW39
		NW40	NW41	NW42	NW43	NW44	
WLF	Create a dynamic website that incorporates a scripting language (client- or server-side).	WL12	WL64	WL74	WL76	WL77	WL78
		WL79	WL80	WL81	WL82	WL83	WL84
		WL85	WL86	WL87	WL88	WL89	WL90
		WL91	WL92	WL95	WL97	WL93	WL94

WLD	Create a dynamic website that incorporates a user-friendly design.	WL33	WL34	WL42	WL43	WL44	WL45
		WL46	WL47	WL49	WL50	WL51	WL53
		WL55	WL56	WL64	WL73	WL74	WL91
		WL100	WL101	WL102			
WLG	Create a dynamic website that incorporates generally accepted standards.	WL21	WL22	WL23	WL27	WL28	WL29
		WL30	WL45	WL46	WL73	WL74	WL91
		WL92	WL93	WL94			
WLH	Create a dynamic website that incorporates web standards.	WL12	WL73	WL74	WL99		
IMC	Create a relational data model for a problem.	IM16	IM30	IM32	IM34	IM36	IM38
		IM39	IM40	IM41	IM42	IM51	IM54
		IM55	IM56	IM57	IM58	IM60	IM61
WLI	Create a secure website that accesses a database.	WL12	WL74	WL89	WL91	WL92	WL93
		WL94	WL97	WL99	WL103	WL104	
WLE	Create a website that incorporates XHTML/CSS.	WL12	WL21	WL22	WL23	WL24	WL25
		WL26	WL27	WL28	WL29	WL30	WL31
		WL32	WL33	WL47	WL56	WL57	WL58
		WL74	WL86	WL87	WL88	WL89	WL95
SEG	Create and specify the software design for a medium-sized software project	SE11	SE12	SE13	SE15	SE16	SE18
		SE19	SE20	SE21	SE22	SE23	SE24
		SE25	SE26	SE27	SE28	SE29	SE30
		SE31	SE47	SE72			

ALA	Demonstrate mathematical literacy in the concepts applicable to Algorithms and Data Structures	AL1	AL2	AL3	AL4	AL5	AL6
		AL7	AL8	AL11	AL15	AL16	AL17
		AL18	AL19	AL53	AL54	AL114	AL123
		AL124	AL125	AL126	AL127	AL128	AL131
		AL132	AL133				
SEE	Demonstrate the central elements of team building and team management. (Software Engineering Management)	SE26	SE35	SE40	SE41	SE42	SE43
		SE64					
IMD	Design a normalized database.	IM27	IM28	IM30	IM36	IM38	IM39
		IM44	IM45	IM46	IM47	IM48	IM49
		IM59					
SED	Display competence with enabling technologies for software engineering	SE4	SE7	SE8	SE9	SE32	SE33
		SE34	SE38	SE39	SE46	SE75	SE76
IMF	Embed relational database technology in a programming or web environment.	IM20	IM54	IM55	IM56	IM57	IM58
		IM59					
SEH	Evaluate different designs prepared as solutions to the same problem.	SE11	SE13	SE15	SE16	SE18	SE19
		SE21	SE22	SE23	SE24	SE25	SE26
		SE27	SE28	SE29	SE30	SE31	SE47
		SE72					
WLL	Gather traffic data for use in web analysis.	WL63					

ALS	Given a problem, identify an appropriate algorithm and/or data structure to solve it	AL47	AL48	AL69	AL70	AL71	AL72
		AL99	AL111	AL118	AL147	AL148	AL149
WLA	Given a problem, suggest an internet infrastructure suitable to solve the problem and justify your choice.	WL1	WL2	WL3	WL6	WL7	WL8
		WL11	WL88	WL91	WL92	WL93	WL94
		WL96	WL98	WL102			
NWE	Identify different network devices, and describe transmission media and topologies for combining them into networks.	NW50	NW51	NW52	NW53	NW54	NW56
		NW57	NW58	NW59	NW60	NW61	NW62
HWH	Maintain hardware/operating environment/networking	HW14	HW15	HW16	HW17	HW18	HW19
		HW42					
NWA	Manage a network for optimal performance, and troubleshoot the network.	NW1	NW3	NW4	NW5	NW6	NW7
		NW8	NW9	NW10	NW11	NW12	NW13
		NW14	NW15				
IMI	Normalize relations	IM45	IM46	IM47	IM48	IM49	
WLK	Optimize a website for search engine access (SEO).						
IPS	Perform complexity analysis	IP25	IP78	IP79	IP80	IP81	IP82
		IP83	IP84	IP85	IP86	IP92	
IPE	Perform simple input and output	IP10					

SEF	Produce a set of software requirements for a medium-sized software system (Requirements).	SE55	SE56	SE57	SE58	SE59	SE60
		SE61	SE62	SE63			
HWG	Program at a low level	HW13					
NWH	Provide a basic overview of networks, at the highest level.	NW81					
IPH	Read and explain code	IP1	IP2	IP3	IP4	IP5	IP28
		IP128					
SEC	Select, with justification, an appropriate set of tools to support the development of a particular software product. (Tools and Environments)	SE4	SE7	SE8	SE9	SE32	SE33
		SE34	SE36	SE37	SE38	SE39	SE46
		SE75	SE76				
SEJ	Select, with justification, the software development models and process elements for the development and maintenance of a particular software product	SE43	SE44	SE45	SE46	SE47	SE48
		SE49	SE72				
IPJ	Successfully deal with errors	IP28	IP29	IP30	IP31	IP32	IP65
		IP67	IP71	IP73	IP74		
SEL	Test code with unit tests, system tests, and user tests	SE61	SE65	SE66	SE67	SE68	SE69
		SE71	SE72	SE73	SE74	SE75	SE76

WLJ	Use an appropriate range of tools to create a multimedia website.	WL37	WL38	WL39	WL40	WL41	WL42
		WL45	WL47	WL56	WL57	WL58	WL61
		WL62	WL63	WL65	WL66	WL67	WL68
		WL69	WL70	WL71	WL72	WL73	WL87
		WL99	WL100	WL101			
HWF	Use and maintain an operating system	HW15	HW16	HW17	HW18	HW19	HW45
IPM	Use collections	IP39	IP40	IP41	IP42	IP43	IP44
		IP45	IP46	IP47	IP87	IP88	IP89
		IP91	IP110				
IPD	Use conditional structures	IP4	IP10	IP11	IP12	IP13	IP125
		IP126					
IMJ	Use constraints	IM50	IM51	IM52	IM53		
IPV	Use dynamic programming	IP93					
ALE	Use finite state machines and regular expressions	AL152	AL153	AL154	AL157	AL158	AL159
		AL160					
IPN	Use generics/templates	IP51	IP52	IP53			
ALO	Use graphs	AL134	AL135	AL136	AL137	AL138	AL139
		AL140	AL141	AL142	AL143	AL144	AL145
		AL146					
ALK	Use hashing	AL59	AL60	AL62	AL66	AL67	AL68
		AL71	AL72	AL73	AL74	AL75	
ALN	Use heaps	AL63	AL65	AL66	AL67	AL68	
IPC	Use iteration	IP5	IP10	IP13	IP16	IP17	

IPK	Use language reference materials	IP33	IP34	IP35			
ALF	Use lists	AL35	AL44	AL55	AL56	AL57	
IPG	Use modelling tools and techniques (problem solving)	IP15	IP18	IP19	IP27	IP66	IP90
		IP111	IP112	IP113	IP114	IP115	
ALQ	Use O(n log n) sorts	AL90	AL91	AL92	AL93	AL94	AL95
		AL96	AL97				
ALP	Use O(n squared) sorts	AL81	AL82	AL83	AL84	AL85	AL86
		AL87	AL88	AL89	AL96	AL97	
IPU	Use object-oriented programming	IP103	IP104	IP105	IP106	IP107	IP108
ALJ	Use ordered trees, e.g., binary search trees, B trees, B+ trees	AL107	AL108	AL109	AL110	AL111	
IPF	Use pointers and references	IP49	IP50				
ALM	Use priority queues	AL64	AL65	AL66	AL67	AL68	
IPI	Use procedures, functions, methods, subroutines a.k.a. Top-down design	IP10	IP22	IP23	IP24		
ALH	Use queues and deques	AL37	AL38	AL46	AL55	AL56	
ALC	Use recursion	AL77	AL78	AL79	AL80	AL129	
IPP	Use recursion	IP58	IP59	IP60	IP61	IP67	IP68
		IP69	IP70	IP119			
ALR	Use searching algorithms	AL28	AL29	AL30	AL57	AL58	AL59
WLB	Use services for communication and to access internet-based resources.	WL7	WL8	WL12	WL60	WL61	WL62
		WL66	WL67	WL68	WL69	WL70	WL71
		WL72					
IPL	Use simple datatypes (primitives)	IP36	IP37	IP38	IP124	IP125	
ALG	Use stacks	AL36	AL45	AL55	AL56		
IPO	Use testing, preconditions, postconditions, assertions	IP54	IP55	IP56	IP57	IP62	IP63

		IP64	IP65	IP126			
IPB	Write and test good code in more than one programming language	IP3	IP6	IP7	IP8	IP9	IP10
		IP48	IP72	IP90	IP116	IP120	IP121
		IP122	IP123	IP129	IP130		
IPQ	Write event-driven programs	IP72	IP73				
IPR	Write multi-threaded programs	IP74	IP75				
IME	Write syntactically correct and accurate SQL statements.	IM11	IM20	IM22	IM24	IM26	IM28
		IM29	IM31	IM33	IM35	IM37	IM53

Summary outcome SEA (Complete a team-based project) has no specific enabling outcomes listed. Nonetheless, a student completing this FPM must have completed a team-based project.

Please note that there is some duplication in this table. In particular, the summary outcome “Use finite state machines” is repeated twice, as is “Use recursion”. Ed Knorr has suggested more expansive wordings for these duplications:

> *“Use finite state machines in writing a program involving states and transitions” for Introductory Programming and “Use finite state machines in the design of an algorithm involving states and transitions” for Algorithms and Data Structures.*
>
> *“Use recursion in the design of a non-trivial algorithm” for Algorithms and Data Structures and “Use recursion in the implementation of an algorithm” for Introductory Programming.*

With these clarifications it is clearer that the levels at which students satisfy these outcomes will be lower in Introductory Programming than in Algorithms and Data Structures.

Appendix 5 - The perspective from the registrars

Registrars are not in favour of FPMs.

Here are several perspectives, from the Registrar at Trinity Western University (http://gvmcmillan.wordpress.com/2010/10/19/can-you-touch-your-toes/), and Registrar Office staff at UNBC, UVic, and UBC (all excerpted from emails sent to committee members).

Can You Touch Your Toes?

How flexible are you?

Me? I've never been able to touch my toes – not since I can ever remember. I blame it on my short arms, but it might have something to do with too much muscle… er… well, bulk, anyhow… ahem.

But if I can't be flexible at my waist, I can at least help my university be flexible. Just last week, I attended a fall meeting of the BC Registrars Association where we discussed a relatively new development: flexible pre-major agreements. Have you heard of them? BC Council on Admission and Transfer has some information on their transfer innovations section of their website.

Currently, the only flexible pre-major agreement that has been signed and put into circulation is English, and my own institution (TWU) is one of the signatories on the agreement. Of course, agreements and programs like this raise all sorts of registrarial questions. The Registrars around the table at our meeting last week raised a number of good questions:

- Should we place this information on transcripts?
- Should we admit students based on this information?
- Should we transfer courses and credits based on this information?

We all said "NO!"

Now before you get your shorts in a knot and accuse us of being terribly inflexible, let me explain. We are a group committed to being service agents and we often have student interests ahead of institution interests. So we like ideas like flexible pre-majors. The problems come when we create rules and procedures around these ideas – those rules and procedures often restrict students and institutions in ways that are not helpful and end up making us less flexible.

In this particular case, flexible pre-majors turn out to be rather inflexible. Students who want to complete a flexible pre-major must take a carefully prescribed set of courses at their home university or college and the only thing that is flexible is that they have a number of other schools (agreement signatories) that they can transfer to. If we registrars decided that the flexible pre-major would be listed on transcripts, then that most often means these agreements would have to go through a whole other level of program approval at our respective Senates/Councils (read: slow, more regulations, less flex). If we decided that they would be a basis for admission, we're talking more approvals at Senate/Councils (read: slow, more red tape, less flex). All of the above would require more administration, more cost, less service, less flexibility. For example, if we admit students on the basis of a flexible pre-major, what happens if they change their minds? Would we ever not admit a student because the program is full? Ugh, I don't like where this is going at all – not very flexible!

Instead, we said do whatever you want, but we'll just transfer courses as usual, we'll admit students as usual, and we'll let the English department determine if the students meet the flexible pre-major requirements. After all, they created the program – they should be the ones to determine how it works. And if students want to do their own thing – that's up to them. Far be it from us to regulate their lives that much.

Now pardon me while I go back to trying… to… touch… my… toes…(just a minute, maybe if I bob up and down a bit)…ok, maybe if I loosen my belt a notch…(hang on, I can do this)… maybe if I bend my knees and curl my toes up… YAY! I did it!

You might say I cheated: "You can't do that!" I'll just say I have flexible toes and flexible rules

Comments from Other Registrars

From UNBC:

…

Thank you for the info from BCCAT. I recognize the intent of the flexible pre-majors; however, in discussion with the Admissions staff, it's not something that we've treated differently than other transfers. The main hurdles being that many transfer institutions do not identify their students as being in "Flexible Pre-Major" programs, so we have no way of knowing they're enroled [sic] in a Flexible Pre-Major and therefore admit students as we would any other BC Transfer Student.

We also don't admit directly to the majority of our programs, so the flexible pre-major would need to be addressed at the Advising level upon declaration of major (unless we determine transfer based on an "Intended" program of study, which creates other issues if a student changes their mind).

Having said all of this, we could provide you with a grid of 100 & 200 Level Computer Science requirements and how courses from sending institutions across the province are received by UNBC. From what I can see, there is similar grid already in place for Mathematics flexible pre-majors.

To sum up, if an institution transcripts their student as being in a Flexible Pre-Major we would admit them to UNBC as a BC College Transfer student, assign appropriate transfer credit (as per the BC Transfer Guide), and ensure that the credit is applied appropriately at the time the student declares their major, as per any signed pre-major agreement.

From UVic:

(1) At present here at UVic we don't know if a transferring student has completed an FPM because -- of course! -- it doesn't appear on the transcript. Our Associate Registrar (who replied to my queries) is considering requesting that the question be asked on the online application for admission (i.e., self-declaration).

(2) Since the FPM doesn't appear on a transcript, the student is handled like any other student applying for admission. They would have enough credits to be considered for transfer and would have to meet the current GPA cutoff for transfer students.

My understanding from Kathleen Boland (AR here at UVic) is that FPMs have been discussed amongst registrars at the provincial level. That's how the BCCAT working group on FPMs learned that there is currently little desire amongst registrars to add FPMs as a transcript annotation.

From UBC:

…The Admission Advisors that I spoke to had never seen [an FPM designation on a transcript]. Even if there was one on the transcript, it would be ignored as it is information that does not factor into the admission process. It would not be noted or recorded anywhere on the student's record. As we admit to degree program only and not to a major, this information does not currently have a function for us. I am assuming that students who have completed an FPM would still have completed courses that individually would transfer over as transfer credit. In which case, I am again assuming that when they eventually do apply to the major, someone in the department must review their file and somehow determine that they have completed the FPM. A question that came to mind is what would happen if a course in the FPM did not transfer over. However maybe that does not matter as much as we are used to as the more important piece of information is that the student has completed the FPM and therefore met the specific requirements here at UBC.

…[W]e would not (also, we would not be able to) give any preference to FPM students. Admission criteria are general to the degree program overall and must be consistently applied so we would not be able to give preference to a specific group of students unless there was a formal arrangement, such as a degree partnership or bridging program. Any kind of arrangement where different admission criteria are to be applied would have to go through the formal Senate processes leading to publication in the academic calendar.

Appendix 6 – Enabling Outcomes – Algorithms and Data Structures

AL1	Graph the following functions: c, lg x, x, x lg x, x^2, 2^x.
AL2	Given a problem statement, describe a solution using sets, functions, and mathematical symbols.
AL3	Use mathematical notation and constructs to describe a problem.
AL4	Apply sets and functions to solving computing problems; for example: hashing, complexity analysis, and counting.
AL5	Translate general problems into rigorous problem statements using set terminology and notation.
AL6	Define the term "mapping".
AL7	Define a mapping between sets.
AL8	Prove *one-to-one* and *onto* for finite and infinite sets.
AL9	Define "time complexity".
AL10	Define "space complexity".
AL11	Classify the different functions in terms of their complexity; for example: c (constant), lg x (logarithmic), x (linear), x lg x, x^2, 2^x (exponential).
AL12	Given a code fragment, identify its time and/or space complexity.
AL13	Given a code fragment, derive its time and/or space complexity.
AL14	Compare and contrast code fragments based on their time and/or space complexity.
AL15	Explain asymptotic behaviour.
AL16	Define "Big-O".
AL17	Define "Big-Omega".
AL18	Define "Big-Theta".
AL19	Compare and contrast Big-O, Big-Omega, and Big-Theta notations.
AL20	Use complexity to estimate the time taken to execute code fragments.
AL21	List the program operations which affect efficiency/complexity (e.g., number of instructions, steps, function calls, comparisons, swaps).
AL22	Given a code fragment, identify the dominant program operations.
AL23	Define *input size* for a given algorithm.
AL24	Determine the effect (in terms of performance) that input size has on an algorithm.
AL25	Give examples of practical limits of algorithms considering complexity.
AL26	Explain the differences between best-, worst-, and average-case complexity analysis.
AL27	Describe why best-case complexity analysis is rarely relevant and how worst-case complexity analysis may never be encountered in practice.
AL28	Given a list and a target, explain how the sequential search attempts to find the target.
AL29	Recall the Big-O value for a sequential search.
AL30	Derive the Big-O value for a sequential search.
AL31	Given an algorithm, compute its worst-case asymptotic complexity.

AL32	Define the term "abstraction".
AL33	Define the term "implementation".
AL34	Differentiate between an abstraction and an implementation.
AL35	Describe list data structures along with their public-interface specifications.
AL36	Describe stack data structures along with their public-interface specifications.
AL37	Describe queue data structures along with their public-interface specifications.
AL38	Describe deque/dequeue data structures along with their public-interface specifications.
AL39	Demonstrate how explicit dynamic memory management is handled in [an imperative language] (e.g., allocation, deallocation or garbage collection, memory heap, run-time stack).
AL40	Demonstrate how implicit dynamic memory management is handled in [an imperative language] (e.g., allocation, deallocation or garbage collection, memory heap, run-time stack).
AL41	Use pointers/references in [an imperative language].
AL42	Describe the advantages and disadvantages of using pointers/references.
AL43	Describe the risks of using pointers/references (e.g., dangling pointers, memory leaks).
AL44	Implement list data structures using both index-based and reference/pointer techniques.
AL45	Implement stack data structures using both index-based and reference/pointer techniques.
AL46	Implement queue data structures using both index-based and reference/pointer techniques.
AL47	Provide examples of problems that can be solved using stacks, queues, and deques.
AL48	Given a problem, solve it using an appropriate choice of stacks, queues, and deques.
AL49	Define the term "iteration".
AL50	Define the term "recursion".
AL51	Recognize algorithms as being iterative or recursive.
AL52	Define the term "loop invariant".
AL53	Prove that a loop invariant holds for a given code fragment.
AL54	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).
AL55	Implement iterative and recursive versions of operations on list, stack and queue data structures.
AL56	Compare and contrast iterative and recursive versions of operations on list, stack and queue data structures.
AL57	Given a sorted list and a target, explain how binary search attempts to find the target.
AL58	Provide an appropriate Big-O estimate for binary search.
AL59	Given a hash table, hash function and target, explain how the hash search attempts to find the target.
AL60	Provide an appropriate Big-O estimate for a hash search.

AL61	Describe tree data structures along with their public-interface specifications.
AL62	Describe hash-table data structures along with their public-interface specifications.
AL63	Describe heap data structures along with their public-interface specifications.
AL64	Describe priority-queue data structures along with their public-interface specifications.
AL65	Implement and manipulate a heap using an index-based technique.
AL66	Implement tree, hash-table, heaps and priority-queue data structures using both index-based and reference/pointer techniques.
AL67	Implement iterative and recursive versions of operations on tree, hash-table, heaps and priority-queue data structures.
AL68	Compare and contrast iterative and recursive versions of operations on tree, hash-tables, heaps and priority-queue data structures.
AL69	Given a problem, describe how (and if) it could benefit from an appropriate choice of priority queues, heaps, and trees.
AL70	Provide examples of problems that can benefit from an appropriate choice of priority queues, heaps, and trees.
AL71	Provide examples of the types of problems that can benefit from a hash data structure.
AL72	Compare and contrast open addressing and chaining for hash data structures.
AL73	Evaluate collision resolution policies for hash data structures.
AL74	Describe how hashing can degenerate from O(1) expected complexity to O(n).
AL75	Identify the types of search problems that do not benefit from hashing (e.g., range searching) and explain why.
AL76	Define the term "tail recursion".
AL77	Describe the benefits of recursion.
AL78	Describe the benefits of tail recursion.
AL79	Draw a recursion tree and relate its depth to a) the number of recursive calls and b) the size of the runtime stack.
AL80	Indicate whether or not a given recursive code fragment terminates.
AL81	Given an input list and a comparison function, sort the list using bubble sort.
AL82	Provide an appropriate Big-O estimate for bubble sort.
AL83	Implement bubble sort.
AL84	Given an input list and a comparison function, sort the list using selection sort.
AL85	Provide an appropriate Big-O estimate for selection sort.
AL86	Implement selection sort.
AL87	Given an input list and a comparison function, sort the list using insertion sort.
AL88	Provide an appropriate Big-O estimate for insertion sort.
AL89	Implement insertion sort.
AL90	Given an input list and a comparison function, sort the list using merge sort.
AL91	Provide an appropriate Big-O estimate for merge sort.
AL92	Implement merge sort.

AL93	Given an input list and a comparison function, sort the list using quicksort.
AL94	Provide an appropriate Big-O estimate for quicksort.
AL95	Implement the quicksort algorithm.
AL96	Compare and contrast the space requirements for different sorting algorithms.
AL97	Compare and contrast the time complexity for sorting algorithms.
AL98	State differences in performance for large datasets versus small datasets on various sorting algorithms.
AL99	For a given scenario, choose an appropriate sorting algorithm and justify your choice.
AL100	Define the term "tree".
AL101	Define and/or describe a binary tree.
AL102	Apply basic tree definitions to classification problems.
AL103	Explain why a binary tree is useful.
AL104	Present an algorithm to find the height of a binary tree.
AL105	Discuss tree traversal algorithms - InOrder, PostOrder, PreOrder, LevelOrder
AL106	Discuss the Big-O values of InOrder, PostOrder, PreOrder, and LevelOrder traversal algorithms.
AL107	Explain why a binary search tree is useful in CS.
AL108	Present common binary search tree algorithms such as search for data, adding data, deleting data.
AL109	Discuss the Big-O value of common binary-search tree algorithms (search for data, adding data, deleting data).
AL110	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].
AL111	Compare and contrast ordered versus unordered trees in terms of complexity and scope of application.
AL112	Categorize an algorithm into one of the common complexity classes (e.g., constant, logarithmic, linear, quadratic, etc.).
AL113	Given two or more algorithms, rank them in terms of their time and space complexity.
AL114	Compare and contrast [the concepts of] space and time complexity.
AL115	Describe the structure, navigation and complexity of an order m B+ tree.
AL116	Insert and delete elements from a B+ tree.
AL117	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.
AL118	Give examples of the types of problems that B+ trees can solve efficiently.
AL119	Compare and contrast B+ trees and hash data structures.
AL120	Explain why B+ trees are preferred dynamic data structures in relational database systems.
AL121	Discuss the trade-offs 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.

AL122	Given a [program fragment], write a formula which computes the number of steps executed as a function of the size of the input (N).
AL123	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).
AL124	In simpler cases, determine the loop invariant.
AL125	Apply counting principles to determine the number of arrangements or orderings of discrete objects, with or without repetition, and given various constraints.
AL126	Use appropriate mathematical constructs to express a counting problem (e.g., counting passwords with various restrictions placed on the characters within).
AL127	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.
AL128	Solve problems using combinatorial arguments and algebraic proofs.
AL129	State the relationship among recursion, Pascal's Triangle, and Pascal's Identity.
AL130	Define the term "binomial distribution" and identify applications in which binomial distributions arise.
AL131	Model and solve appropriate problems using binomial distribution.
AL132	Apply basic probability theory to problem solving, and identify the parallels between probability and counting.
AL133	Define various forms of the pigeonhole principle; recognize and solve the specific types of counting and hashing problems to which they apply.
AL134	Discuss the BigO of spanning-tree algorithms.
AL135	Perform breadth-first and depth-first searches in graphs.
AL136	Explain why graph traversals are more complicated than tree traversals.
AL137	Discuss Prim's and Kruskal's minimal spanning-tree algorithms.
AL138	Discuss the Big-O of minimal spanning-tree algorithms.
AL139	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
AL140	Prove basic theorems about simple graphs (e.g., handshaking theorem).
AL141	Explain the computer representation of graphs.
AL142	Convert between adjacency matrices / lists and their corresponding graphs.
AL143	Determine whether a given graph is a subgraph of another.
AL144	Discuss the complexity of the Travelling Salesman problem
AL145	Explain Dijkstra's Algorithm for the Shortest Path in a graph
AL146	Discuss the Big-O of Dijkstra's algorithm
AL147	Apply object oriented and modular design techniques to an application problem to design a software solution.
AL148	Given a problem, select the most appropriate data structure (lists, stacks, queues, trees, hash tables, heaps, priority queues) for its solution.

AL149	Implement an application design, including an implementation an appropriate data structure (lists, stacks, queues, trees, hash tables, heaps, priority queues).
AL150	Analyze [imperative-language] programs and functions to determine their algorithmic complexity.
AL151	Given a code fragment, trace its operation by hand.
AL152	Discuss the concept of finite state machines.
AL153	Discuss the concept of a deterministic finite automaton.
AL154	Explain context-free grammars.
AL155	Define the term "finite state machine".
AL156	Define the term "regular expression".
AL157	Given a problem which can be solved by using a regular expression, create the regular expression.
AL158	Design a deterministic FSM to accept a simple regular expression.
AL159	Explain how some problems have no algorithmic solution.
AL160	Provide examples that illustrate the concept of uncomputability.

Appendix 7 – Enabling Outcomes – Computer Architecture

CA1	Define the term “computer system architecture”.
CA2	Define the term “computer architecture”.
CA3	Describe the progression of computer architecture from vacuum tubes to VLSI.
CA4	List and describe the fundamental building blocks of a computer system.
CA5	List and describe the fundamental building blocks of a computer processor.
CA6	For each of the fundamental computer-system building blocks, explain its role in the historical development of computers.
CA7	For each of the fundamental computer-processor building blocks, explain its role in the historical development of computers.
CA8	Use mathematical expressions to describe the functions of simple combinational circuits.
CA9	Use mathematical expressions to describe the functions of simple sequential circuits.
CA10	Design a simple circuit using the fundamental building blocks.
CA11	Justify the different formats used to represent numerical data (e.g., floating point, integer).
CA12	Enumerate/compare and contrast the different formats used to represent numbers in a processor.
CA13	Explain how integers are stored in sign-magnitude representation.
CA14	Explain how integers are stored in twos-complement representation.
CA15	Trace the numeric operations involved in performing add and subtract on twos complement numbers.
CA16	Convert among binary, octal, decimal and hexadecimal number formats.
CA17	Define the term “accuracy”.
CA18	Define the term “precision”.
CA19	Discuss how fixed-length number representations affect accuracy and precision.
CA20	Discuss the differences in the internal representations of numeric vs. non-numeric data (with respect to a specific programming environment).
CA21	Describe the internal representation of characters.
CA22	Describe the internal representation of strings.
CA23	Describe the internal representation of records.
CA24	Describe the internal representation of arrays.
CA25	Explain the organization of the classical von Neumann machine and its major functional units.
CA26	Explain the fetch-decode-execute-update cycle involved in instruction processing.
CA27	Compare and contrast how instructions are represented at both the machine level and in the context of a symbolic assembler.

CA28	Compare and contrast different instruction formats, such as addresses per instruction and variable length vs. fixed-length formats.
CA29	Trace simple assembly-language program segments and their effect on registers, memory, and the program counter.
CA30	Write simple assembly language program segments to meet provided specifications.
CA31	Demonstrate how fundamental high-level programming constructs are implemented at the machine-language level.
CA32	Explain how subroutine calls are handled at the assembly level.
CA33	Explain the role of a stack in a subroutine call (i.e., local variables, save registers, return address, etc.)
CA34	Trace the effect of a subroutine call on registers, memory, and the program counter.
CA35	Trace the effect of a subroutine return on registers, memory, and the program counter.
CA36	Define the term "interrupt".
CA37	Describe the use of interrupts ~~and~~ with I/O operations.
CA38	Explain the relationship between interrupts and I/O operations.
CA39	Trace the effect of an interrupt call on registers, memory, and the program counter.
CA40	Compare and contrast the main types of memory technology.
CA41	Define the term "memory latency".
CA42	Explain the effect of memory latency on instruction execution time.
CA43	Describe the memory hierarchy (registers, caches, main memory, flash memory, magnetic disk, network storage devices, cloud devices, etc.)
CA44	Explain how the use of a memory hierarchy could reduce effective memory latency.
CA45	Describe the motivation for memory management.
CA46	Describe the principles of memory management.
CA47	Describe the role played by a hardware cache in the memory hierarchy.
CA48	Define the term "virtual memory".
CA49	Compare and contrast virtual memory with physical memory.
CA50	Explain the workings of a simple memory management system (e.g., address translation, memory allocation for a program).
CA51	Explain how interrupts & interrupt service routines co-operate to implement I/O control and data transfers.
CA52	Define the term "bus".
CA53	Identify various types of buses in a computer system.
CA54	Describe the role played by the various system buses.
CA55	Trace the path taken by data accessed from a magnetic disk drive to main memory.

CA56	Describe the role played by software components involved in the transfer of data to and from a disk drive.
CA57	Compare the common network configurations.
CA58	Identify hardware interfaces and other hardware extensions suitable for multimedia support.
CA59	Describe the advantages and limitations of RAID architectures (e.g., latency, reliability).
CA60	Describe the datapath within a non-pipelined microprocessor architecture.
CA61	Compare and contrast alternative implementations of datapaths.
CA62	Define what is meant by “control point” and “control signal” within a non-pipelined microprocessor architecture.
CA63	Discuss the concept of control points and the generation of control signals using hardwired or microprogrammed implementations.
CA64	Define the term “parallelism”, in general terms.
CA65	Define the term “instruction-level parallelism”.
CA66	Identify when and where major hazards occur in a pipeline.
CA67	Describe how pipelining achieves instruction-level parallelism.
CA68	Describe the concept of parallel processing beyond the classical von Neumann model.
CA69	Compare and contrast alternative architectures such as SIMD, MIMD (e.g., GPUs, customized devices).
CA70	Compare and contrast interconnection networks and characterize their different approaches.
CA71	Define the term “multiprocessing”.
CA72	Discuss the special concerns that multiprocessing systems present with respect to memory management and describe how these are addressed (e.g., memory consistency, cache coherency, effect on system software, NUMA, etc.).
CA73	Define the term “multithreading”.
CA74	Describe how multithreading can achieve performance improvement.
CA75	Explain the factors that can prevent the performance advantages multithreading can offer.
CA76	Define the term “scalability” in the context of computer-system performance (e.g., memory, disk space, processors, etc.).
CA77	Discuss the hardware-resource constraints that limit scalability.
CA78	Compare and contrast LANs and WANs.
CA79	Describe the physical organization of a network (e.g., the Internet).
CA80	Discuss the software architecture issues involved in the design/implementation of a layered network protocol.
CA81	Explain how architectures differ in network and distributed systems.
CA82	Discuss the software architecture and performance issues related to network-based vs. local computing/applications.

CA83	Enumerate performance metrics.
CA84	Describe several performance benchmarks and what they measure.
CA85	Compare and contrast alternate performance benchmarks.
CA86	Analyze the claims made in performance reports (e.g., magazine articles, web pages).

Appendix 8 – Enabling Outcomes – Hardware

HW1	Identify and categorize different types of computers, e.g., mini, micro, laptop, smartphone, etc.
HW2	Identify the components of computers and peripherals.
HW3	Describe the functions and roles of computer components and peripherals.
HW4	Explain how the computer components are connected.
HW5	Explain how the computer components communicate to accomplish different tasks.
HW6	Describe the internal structure of the Central Processing Unit, including multicore processors.
HW7	Describe the operation of the Central Processing Unit in terms of instruction execution.
HW8	Explain the hierarchy of memory (e.g., disks, caches, RAM, registers).
HW9	Explain the operation of the hierarchy of memory in terms of program execution.
HW10	Describe the different types of memory (e.g., RAM, ROM, BIOS, video RAM).
HW11	Define the term "virtual memory".
HW12	Explain how virtual memory functions.
HW13	Explain how machine language provides the foundation for all programming languages.
HW14	Install and test basic computer hardware and peripherals.
HW15	Install, configure, and test an operating system.
HW16	Install and test device drivers.
HW17	Perform routine OS, firmware and device driver maintenance.
HW18	Troubleshoot basic computer hardware, OS and device driver problems, demonstrating basic problem solving methodologies.
HW19	Set up and configure a computer for networking connectivity.
HW20	Format, partition and maintain a disk.
HW21	Describe physical disk structure, e.g., sectors, tracks, sides, spindle, etc.
HW22	Explain how factors such as seek time, latency, track density, and RPM, influence disk performance.
HW23	Describe the various methods for encoding data on to a disk.
HW24	Define the term "RAID".
HW25	Describe the standard levels of RAID technology.
HW26	Select the proper RAID level for a specific computing requirement.
HW27	Configure and implement a RAID system according to a specific requirement.

HW28	Identify and explain the purpose of the basic hardware and software necessary with respect to connecting the PC to a network.
HW29	Use appropriate methods to backup and restore system settings.
HW30	Use appropriate methods to backup and restore data.
HW31	Explain the video adapter functions (e.g., chip sets and graphics cards).
HW32	List the key features and functions of a video card.
HW33	Describe how video displays (monitors) work.
HW34	List the key features of a video display (monitor).
HW35	Connect one or more video displays to a computer.
HW36	Describe the work flow of rendering a video image.
HW37	Describe how data is stored to and retrieved from optical discs.
HW38	Describe the different types of solid state memory technologies.
HW39	Describe the different types of random access memories (RAM).
HW40	Compare and contrast solid state drives and hard disks.
HW41	Describe how an audio system works in a personal computer.
HW42	Install and configure an operating system.
HW43	Describe significant technologies that improved CPU performance (e.g., multicore, pipeline).
HW44	Propose a hardware configuration for a specific computing requirement.
HW45	Describe the fetch, increment, and execute cycle.
HW46	Describe how instructions and data are fetched from memory into the CPU.
HW47	Describe the Von Neumann architecture.
HW48	Describe how interrupts and the stack work.
HW49	Identify and explain the use of different ports and connectors on a computer.
HW50	Identify various components on a motherboard, e.g., northbridge and southbridge.
HW51	Describe how the CPU communicates with other devices on the motherboard.
HW52	Describe the various form factors.
HW53	Contrast the performance of different drive interface standards, such as SCSI, SATA, etc.
HW54	Compare and contrast the hardware differences between enterprise, personal, and mobile computing devices.
HW55	Describe how different hardware interface devices work and interact with mice, keyboards, game consoles, etc.
HW56	Describe how touchscreen devices work.
HW57	Describe how computer hardware interfaces influence software interface design.

HW58	Describe the purpose and function of mobile hardware components, including antennae, GPS, accelerometers, Bluetooth, infrared, cameras, SIM cards, etc.
HW59	Compare and contrast the difference in the functional capabilities of laptops, notepads, smartphones, e-readers, and other mobile devices.
HW60	Describe different methods for structuring files on a hard disk.

Appendix 9 – Enabling Outcomes – Information Management

IM1	Define the terms information, information management, metadata, data, database, database management system, metadata, and data mining.
IM2	Compare and contrast metadata, data and information.
IM3	Describe how data, information, and databases are used in organizations.
IM4	Describe how data storage and retrieval has changed over time.
IM5	Compare and contrast the database approach to traditional file processing.
IM6	Describe how the Internet and the demand for information from users outside the organization (customers and suppliers) impacts data handling and processing.
IM7	Define the terms data quality, accuracy and timeliness, and explain how their absence will impact organizations.
IM8	Describe various methods for data collection, such as automated data collection, input forms, data readers, etc.
IM9	Describe basic issues of data retention, including the need for retention, types of media, privacy, security, and legal issues.
IM10	Explain why data backup is important and how organizations use backup and recovery systems.
IM11	Describe the purpose of Structured Query Language (SQL).
IM12	Define the term "relation".
IM13	Define the term "relational database".
IM14	Define the term "table".
IM15	Define the term "attribute".
IM16	List and describe attribute types.
IM17	Describe the purpose and use of a SELECT statement.
IM18	Describe the purpose and use of a WHERE clause.
IM19	Describe the purpose and use of an ORDER BY clause.
IM20	Write and test SQL queries using SELECT, FROM, WHERE, and ORDER BY.
IM21	Describe logical operators (AND, OR, NOT).
IM22	Write and test SQL statements using logical operators.
IM23	Describe set operators (UNION, DISTINCT, LIKE, and BETWEEN).
IM24	Write and test SQL statements using set operators.
IM25	Describe the purpose and use of aggregate functions using GROUP BY and GROUP BY HAVING.
IM26	Write and test SQL statements using aggregate functions with GROUP BY and GROUP BY HAVING.
IM27	Describe the purpose and use of sub-queries, views, and joins.
IM28	Write and test SQL statements using use sub-queries, views, and joins.

IM29	Format output using headers, footers, totals, and subtotals.
IM30	Describe the purpose and use of the CREATE TABLE command.
IM31	Write and test SQL statements using CREATE TABLE.
IM32	Describe the purpose and use of the CREATE VIEW command.
IM33	Write and test SQL statements using CREATE VIEW.
IM34	Describe the purpose and use of the SELECT AS command.
IM35	Write and test SQL statements to create tables and views using SELECT AS.
IM36	Describe the purpose and use of INSERT, UPDATE and DELETE.
IM37	Write and test SQL statements using INSERT, UPDATE and DELETE.
IM38	Describe the purpose and use of *query by example.*
IM39	Write and test a query using query by example.
IM40	Describe the features of the relational model including relations, tuples, attributes, domains and operators.
IM41	Describe the purpose and use of select, project, union, intersection, set difference, cross-product, and natural join relational operations.
IM42	Demonstrate select, project, union, intersection, set difference, cross-product, and natural join relational operations using simple example relations provided.
IM43	Define the terms key, primary key, and foreign key.
IM44	Define the term “functional dependency”.
IM45	Explain the relationship between functional dependencies and keys and give examples.
IM46	Explain how having normal form relations reduces or eliminates attribute redundancy and update/delete anomalies.
IM47	Normalize a set of relations to third normal form.
IM48	Normalize a set of relations to Boyce-Codd normal form.
IM49	Normalize a set of relations to fourth normal form.
IM50	Define and explain the need for referential integrity.
IM51	Explain the primary key requirements for referential integrity.
IM52	Describe the purpose and use of constraints.
IM53	Write and test user-defined integrity constraints.
IM54	Describe the purpose and use of Entity Relationship and UML data modelling diagrams.
IM55	Define the term “cardinality”.
IM56	Use cardinality notation in an Entity Relationship or UML diagram.
IM57	Given an Entity Relationship or UML diagram, interpret the diagram.
IM58	For a given scenario, create Entity Relationship and UML data modelling diagrams.
IM59	For a given scenario, design a normalized relational database.
IM60	Describe the relationship between a logical model and a physical model.

IM61	Describe the use of CASE tools in data modelling.
IM62	Describe the purpose and use of a data warehouse.
IM63	Compare and contrast data administration and database administration.
IM64	Describe issues in database security.

Appendix 10 – Enabling Outcomes – Introductory Programming

IP1	Given a code fragment, describe its purpose in plain English.
IP2	Given a code fragment, trace its execution.
IP3	Adapt an existing code fragment to change its behaviour.
IP4	Modify conditional structures in a short program.
IP5	Modify iterative structures in a short program.
IP6	Write well-structured, well-documented, well-commented readable code.
IP7	Describe the role of documentation and comments.
IP8	Use language-appropriate idioms.
IP9	Write meaningful, well-structured external documentation.
IP10	Design, implement, test, and remove errors from a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, basic conditional and iterative structures, and functions.
IP11	Describe the syntax and semantics of conditional structures available in [a language].
IP12	Use conditional structures available in [a language].
IP13	Choose appropriate conditional and/or iterative constructs for a given programming task, and justify your choice.
IP14	Define the term “pseudocode”.
IP15	Use pseudocode and/or diagrams to describe the steps involved in solving simple problems.
IP16	Describe the syntax and semantics of iteration structures available in a language.
IP17	Use iterative structures available in a language.
IP18	Apply decomposition techniques to break a program into smaller pieces (where each piece has a specific purpose or responsibility).
IP19	Explain the role of pseudocode and diagramming in decomposing problems.
IP20	Define the term “formal parameter”.
IP21	Define the term “actual parameter”.
IP22	Given a code fragment, identify formal and actual parameters of a function.
IP23	Describe the role of formal and actual parameters of a function.
IP24	Identify function.
IP25	List various desirable properties of an algorithm.
IP26	Define the term “algorithm”.
IP27	Given a simple problem, create an algorithm to solve it.
IP28	Trace the execution of a program (e.g., desk checking).
IP29	Describe and use strategies for removing syntax errors.
IP30	Describe and use strategies for removing logic errors.
IP31	Describe and use strategies for removing runtime errors.

IP32	Interpret error messages (compiler, run-time, etc.) and identify their causes.
IP33	Use appropriate sources to learn about the programming environment.
IP34	Apply appropriate sources to aid in the building of programs.
IP35	Explore language features using authoritative language documentation.
IP36	Discuss the representation and use of primitive data types.
IP37	Discuss the representation and use of built-in data structures (e.g., strings, arrays, files).
IP38	Describe how to allocate, manipulate, and use strings.
IP39	Describe how to allocate, manipulate and use arrays.
IP40	Describe how to allocate, manipulate, and use records.
IP41	Describe how to allocate, manipulate and use lists, stacks, and queues.
IP42	Describe how to allocate, manipulate, and use trees.
IP43	Describe how to allocate, manipulate, and use graphs.
IP44	Describe how to allocate, manipulate, and use hash tables.
IP45	Implement user-defined data structures in a high-level language.
IP46	Compare alternative implementations of data structures with respect to performance, both time and space.
IP47	Compare and contrast dynamic and static data structure implementations.
IP48	Choose an appropriate data structure for modelling a given problem.
IP49	Use pointers/references to implement user-defined data structures.
IP50	Implement user-defined data structures containing pointers/references.
IP51	Use existing generics/templates to solve a given problem.
IP52	Write a generic function to generalize the solution to a given problem.
IP53	Demonstrate familiarity with contents of industry-standard libraries.
IP54	Create a comprehensive suite of unit tests for a piece of software.
IP55	Critique an existing suite of tests for a piece of software.
IP56	Write appropriate pre- and post-conditions for methods or functions.
IP57	Write appropriate assertions for code fragments.
IP58	Describe the concept of recursion and give examples of its use.
IP59	Given a recursively-defined problem, identify its base case(s) and general case(s).
IP60	Compare and contrast iterative and recursive solutions for elementary problems such as factorial.
IP61	Compare and contrast mathematical induction and recursion.
IP62	Formulate loop invariants for simple loops.
IP63	Demonstrate code correctness given a loop invariant.
IP64	Demonstrate loop termination.
IP65	Demonstrate correct handling of boundary conditions.
IP66	Describe the divide-and-conquer approach.
IP67	Implement, test, and remove errors from simple recursive functions and

	procedures.
IP68	Describe how recursion can be implemented using a stack.
IP69	Discuss problems for which backtracking is an appropriate solution.
IP70	Determine when a recursive solution is appropriate for a problem.
IP71	Develop code that responds to exception conditions raised during execution.
IP72	Explain the differences between event-driven programming and command-line programming.
IP73	Design, code, test and remove errors from simple event-driven programs that respond to user events.
IP74	Design, code, test and remove errors from simple multi-threaded programs.
IP75	Determine when a multi-threaded solution is appropriate for a problem.
IP76	Explain the use of Big-O, Big-Omega, and Big-Theta notation to describe the behaviour of functions.
IP77	Define the term "time complexity".
IP78	Define the term "space complexity".
IP79	Use Big-O, Big-Omega, and Big-Theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms.
IP80	Determine the time and space complexity of simple algorithms.
IP81	Relate the complexity class of an algorithm to its scalability.
IP82	Describe the kinds of operations we can measure in evaluating the performance of an algorithm.
IP83	Rank algorithms by rate of growth.
IP84	Compare and contrast best-, worst- and average-case behaviours.
IP85	Implement a greedy algorithm to solve an appropriate problem.
IP86	Implement the most common quadratic and O (N log N) sorting algorithms.
IP87	Design and implement an appropriate hashing function for an application.
IP88	Design and implement a collision-resolution algorithm for a hash table.
IP89	Discuss the computational efficiency of the principal algorithms for sorting, searching and hashing.
IP90	Discuss factors other than computational efficiency that influence the choice of algorithms, such as program development time, maintainability, and the use of application-specific patterns in the input data.
IP91	Solve problems using fundamental graph algorithms.
IP92	Justify the choice of algorithms for a given problem with reference to algorithm time and space properties.
IP93	Design and implement a dynamic programming solution to a problem.
IP94 – IP102	These enabling outcomes were moved to Algorithms and Data Structures but subsequent enabling outcomes were not renumbered.
IP103	Explain the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance and polymorphism.
IP104	Design, implement, test and debug programs in an object-oriented

	programming language.
IP105	Describe how the class mechanism supports encapsulation and information hiding.
IP106	Design, implement, and test the implementation of is-a relationships among objects using a class hierarchy and inheritance.
IP107	Compare and contrast the notions of overloading and overriding methods in an object-oriented language.
IP108	Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
IP109	Define the term “iterator”.
IP110	Use iterators to access the elements of a container/collection.
IP111	Interpret UML class diagrams.
IP112	Given a problem statement, apply a standard technique to identify the classes involved.
IP113	Create a UML class diagram that associates classes identified in a problem.
IP114	Create a UML sequence diagram representing object interaction.
IP115	Interpret UML interaction diagrams.
IP116	Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.
IP117	Describe the phases of program translation from source code to executable code and the files produced by these phases.
IP118	Explain the differences between machine-dependent and machine-independent translation and where these differences are evident in the translation process.
IP119	Translate a simple iterative construct, such as summing an array or computing a factorial using a loop, into a recursive functional (non-side effecting) construct.
IP120	Describe the strengths and weaknesses of various programming languages.
IP121	Given a problem, indicate an appropriate programming language in which to implement a solution.
IP122	List some good programming standards and practices.
IP123	Use programming standards and practices to create good code.
IP124	Describe Boolean values and operations.
IP125	Given a Boolean expression, evaluate it.
IP126	Given a problem involving conditions, implement and test appropriate Boolean expression(s).
IP127	Define the term “scope”.
IP128	Given a code fragment, identify the scope of the variables involved.
IP129	Write correct non-trivial programs in two or more programming languages.
IP130	Map language-specific terms in one programming language to their equivalents in other languages.

Appendix 11 – Enabling Outcomes – Networking

NW1	Manage networked accounts on a server.
NW2	Define the term “network performance”.
NW3	Measure network performance.
NW4	List and explain the factors that constrain network performance.
NW5	List and explain various techniques to enhance network performance.
NW6	Given a network configuration, recommend an appropriate technique to improve network performance.
NW7	Protect servers from data loss and describe how to recover from data loss.
NW8	Given a network configuration, recommend and implement a server backup and recovery plan.
NW9	Discuss the costs and benefits of network management and planning.
NW10	Develop standards, policies, procedures and documentation for a network.
NW11	Describe the main challenges faced in an organization using networks.
NW12	Explain a systematic approach for troubleshooting a network.
NW13	Implement a systematic approach for troubleshooting a network.
NW14	Troubleshoot a network following a structured approach.
NW15	Describe the types of specialized equipment and other resources available for troubleshooting.
NW16	Explain the OSI reference model.
NW17	Explain the OSI reference model's layers and their relationships to networking hardware and software.
NW18	Discuss the layered architecture of protocols, and describe common protocols and their implementation.
NW19	Describe, compare, and contrast the major network architectures, including TCP/IP.
NW20	Outline the limitations, advantages, and disadvantages of each standard or architecture.
NW21	Define the term “network services”.
NW22	Compare and contrast centralized and client/server computing.
NW23	Define the term “client/server networks”.
	Define the term “peer-to-peer”. Recorded as WL10.
NW24	Discuss the basics of Web-based computing environments.
NW25	Describe the basic concepts associated with wide area networks (WANs).
NW26	Describe how to use the Internet for a private connection using VPNs.
NW27	Describe how to implement a VPN.
NW28	List the pros and cons of VPN.
NW29	Describe virtual LANs.
NW30	Describe how to implement a virtual LAN.

NW31	List the pros and cons of using VLANs.
NW32	Define the term “physical network topology”.
NW33	Define the term “logical network topology”.
NW34	List and describe the basic steps required for network operating system installation.
NW35	Install and configure network applications.
NW36	Create a network security plan.
NW37	Describe WAN protocols, and software and hardware technologies to build WANs.
NW38	Design a small local area network.
NW39	Build a small local area network.
NW40	Maintain a small local area network.
NW41	Describe the process of setting up peer-to-peer networks.
NW42	Set up a peer-to-peer network.
NW43	Describe Frame Relay.
NW44	List commands to monitor Frame Relay operation in the router.
NW45	Compare and contrast Local, Metropolitan and Wide Area Networks.
NW46	Define the following basic networking terms: Client, Peer, Server, the Network Medium, Network Protocol, Network Software, Network Service.
NW47	Describe the basic Network Types: Peer-to-Peer, Server-Based, Personal Area Networks (PANs), Hybrid Networks. Describe Storage-Area Networks (SANs), Server Hardware Requirements, Specialized Servers.
NW48	Define technical terms related to cabling, including attenuation, crosstalk, shielding, and plenum.
NW49	Define the following terms: hub, and switch.
NW50	Describe the basic types of Hubs: Active Hubs, Passive Hubs, Hybrid Hubs.
NW51	Identify major types of network cabling.
NW52	Identify-major types of wireless network technologies.
NW53	Given a particular LAN/WAN environment, identify and justify the appropriate cabling and connectors.
NW54	Explain how network adapters prepare data for transmission, accept incoming network traffic, and control how networked communications flow.
NW55	Define the following terms: repeater, bridge, router, brouter, gateway, and switch.
NW56	Explain how larger networks may be implemented using devices such as repeaters, bridges, routers, brouters, gateways, and switches.
NW57	Configure routers to connect different types of LANs and WANs using LAN and WAN protocols.
NW58	Describe the advantages and methods of network segmentation.
NW59	Name and describe two switching methods.
NW60	Describe full- and half-duplex Ethernet operation.
NW61	Describe the features and benefits of Fast Ethernet.

NW62	Describe the characteristics of various transmission media.
NW63	Compare and contrast base band and broadband transmission technologies.
NW64	Describe rudimentary signaling technologies for mobile computing.
NW65	Explain the IEEE 802 networking model and related standards.
NW66	Describe the function and structure of packets in a network, and analyze them.
NW67	Explain the function of protocols in a network (e.g., TCP/IP).
NW68	Describe various channel access methods, compare and contrast them.
NW69	Discuss the different types of carriers used for long-haul network communications.
NW70	Identify virtual LANs, LAN switching, Fast Ethernets, Frame Relay, ISDN networking.
NW71	Identify the uses, benefits, and drawbacks of advanced WAN technologies such as ATM, FOOI, SONET, and SMDS.
NW72	Describe network congestion problems in Ethernet networks.
NW73	Distinguish between cut-through and store-and-forward LAN switching.
NW74	Describe the operation of the Spanning Tree Protocol and its benefits.
NW75	Compare and contrast the following WAN services: LAPB, Frame Relay, ISDN/LAPD, HDLC, PPP, and DDR.
NW76	Recognize key Frame Relay terms and features.
NW77	Identify PPP operations to encapsulate WAN data on routers.
NW78	Explain and identify key protocol information given samples of captured packets.
NW79	Explain the role of driver software in network adapters.
NW80	Explain the operation fundamentals of network operating systems.
NW81	Provide a basic overview of networks, at the highest level.

Appendix 12 – Enabling Outcomes – Software Engineering

SE1	Define the term “software engineering”.
SE2	Outline the history of software engineering.
SE3	Decompose implementation work into units for parallel implementation.
SE4	Compare and contrast several implementation philosophies (e.g., Big bang, top down, bottom up).
SE5	Define the term “code quality” and describe how it is measured.
SE6	Define the term “productivity” and describe how it is measured.
SE7	Describe and use techniques to improve code quality and productivity by using software tools.
SE8	Describe various aspects of software configuration management.
SE9	Justify the use of software-configuration management (SCM) tools.
SE10	Use an issue tracking system to identify and eliminate problems.
SE11	Design and apply code standards.
SE12	List and describe several good design principles.
SE13	Explain and apply good design principles.
SE14	Define the term “design pattern”.
SE15	Explain and apply common design patterns.
SE16	Select and apply appropriate design patterns in the construction of a software application.
SE17	Define the term “software architecture”.
SE18	Recognize basic software architectures.
SE19	Design and specify a software system's architecture.
SE20	Design and specify the class-level structure of a software system (OO paradigm).
SE21	Design and specify the procedure-level structure of a software system (procedural paradigm).
SE22	Identify the relationships between classes.
SE23	Extend the analysis classes to represent the design use cases and identify specific object instances.
SE24	Add/modify relationships between classes and objects to further extend the design.
SE25	Represent analysis and design models using use case, sequence, collaboration, class, and state machine diagrams.
SE26	Design a project with the UML.
SE27	Design a project in a group setting.
SE28	Describe the qualities of a good software system and explain their value.
SE29	Discuss the properties of good software design including the nature and the role of associated documentation.

SE30	Evaluate the quality of alternative software designs based on key design principles and concepts.
SE31	Measure the size of a project.
SE32	Use feedback from implementation to refine design.
SE33	Analyze and evaluate a set of tools in a given area of software development (e.g., management, modelling, or testing).
SE34	Use a range of software tools in support of the development of a software product of medium size.
SE35	Use 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).
SE36	Define the term “code repository”.
SE37	Define the term “version control system”.
SE38	Coordinate implementation efforts using a code repository.
SE39	List the typical operations provided by a software configuration management (SCM) tool.
SE40	Organize the solution to a medium-sized non-trivial problem involving a group of programmers.
SE41	Apply good project management practices to a software project, including risk analysis, task/resource scheduling, human resource management, and continuous progress monitoring.
SE42	Identify and resolve common team-related issues such as communication problems and decision making.
SE43	Review and evaluate team member performance.
SE44	For each of several software project scenarios, describe the project's place in the software lifecycle, identify the particular tasks that should be performed next, and identify metrics appropriate to those tasks.
SE45	Identify the principal issues associated with software evolution and explain their impact on the software lifecycle.
SE46	Explain the risks of skipping or reducing a phase of the lifecycle.
SE47	Recognize the types of tools that are used in each phase of the software lifecycle.
SE48	Compare and contrast the traditional waterfall development model to the incremental model, the agile model, the object-oriented model, and other common models.
SE49	Apply a software lifecycle model of Object-Oriented paradigm, and its methodology to a multi-member software development project.
SE50	Create good user documentation.
SE51	Discuss the challenges of maintaining software.

SE52	Discuss the challenges of maintaining legacy systems and the need for reverse engineering.
SE53	Define the term “refactoring”.
SE54	Identify weaknesses in a given simple design, and highlight how they can be removed through refactoring.
SE55	Argue for the need for requirements.
SE56	Describe and use techniques for eliciting, analyzing, specifying, and verifying functional and non-functional requirements.
SE57	Describe several types of requirements.
SE58	Elicit requirements from a client.
SE59	Given a narrative, develop an appropriate use case.
SE60	Refine a use case to serve as foundation for design.
SE61	Identify classes based on use cases and narratives.
SE62	Identify the characteristics of good and bad (untestable, ambiguous, unethical) requirements.
SE63	Given a requirement, identify whether it is good or bad, supporting your answer.
SE64	Explain the typical difficulties of technical communication.
SE65	Construct a software test plan.
SE66	Create, evaluate and justify, and implement a test plan for a medium-size code segment.
SE67	Distinguish between the different types and levels of testing (unit, integration, systems and acceptance) for medium-size software products.
SE68	Create test cases.
SE69	As part of a team activity, undertake an inspection of a medium-size code segment.
SE70	Explain basic testing terminology.
SE71	Recognize common testing frameworks employed in the industry.
SE72	Describe and use techniques for verifying and validating all artifacts created during the process.
SE73	Describe and use techniques for testing the resulting system through unit testing, integration testing, system testing, etc.
SE74	Describe and use techniques for implementing user acceptance testing.
SE75	Describe the role that tools can play in the validation of software.
SE76	Write and debug test scripts.

Appendix 13 – Enabling Outcomes – Web Learning

WL1	Describe several major hardware and software components, including how they are related to the Internet infrastructure.
WL2	Describe the roles and importance of TCP/IP in the Internet.
WL3	Describe how the DNS system works.
WL4	Define the term "domain".
WL5	Define the term "top-level domain".
WL6	List several of the different top-level domains and, for each, describe their intended audience(s).
WL7	List several application level protocols (e.g., POP, SMTP, FTP, HTTP).
WL8	Provide examples of how and where application level protocols get used on the Internet.
WL9	Define the term "client/server architecture".
WL10	Define the term "peer-to-peer architecture".
WL11	Compare and contrast client/server and peer-to-peer architectures.
WL12	List and explain several of the considerations when a company hosts a web site (e.g., amount of disk space, monthly transfer limits, etc.).
WL13	Define the term "HTML/XHTML header element".
WL14	Define the term "HTML/XHTML paragraph element".
WL15	Define the term "logical formatting".
WL16	Define the term "list".
WL17	Define the term "table".
WL18	Define the term "HTML/XHTML image element".
WL19	Define the term "HTML/XHTML hyperlink element".
WL20	Define the term "HTML/XHTML character entity".
WL21	Create web pages that conform to W3C standards using headings, paragraphs, logical formatting, lists, tables, images, hyperlinks and character entities.
WL22	Validate and correct a web page.
WL23	Validate and correct a style sheet.
WL24	Differentiate between an absolute and a relative URL or URI and construct the correct one for a given situation.
WL25	Use the appropriate markup to create sections for styling (for example, div and span in XHTML).
WL26	Create tag, pseudo-class, class and id selectors using basic properties (e.g., font, color, text-decoration, text-align, background, list-style-type, etc.).
WL27	Describe Cascading Style Sheets (CSS); that is, what is their purpose
WL28	Describe how Cascading Style Sheets (CSS) are created.
WL29	Create properly-formed CSS style rules.

WL30	Place properly formed CSS rules in an external or internal style sheet.
WL31	Analyze a set of CSS rules for the cascade effect and render the resultant styling to a webpage.
WL32	Describe the CSS box model.
WL33	Use the CSS box model and positioning properties to create a web page with multiple columns, a masthead, and a footer.
WL34	Compare and contrast various web page layouts: liquid, fixed, and jello/elastic.
WL35	Define the term "raster format".
WL36	Define the term "vector format".
WL37	Describe the characteristics of pictures that are best stored in raster format.
WL38	Describe the characteristics of pictures that are best stored in vector formats.
WL39	Compare and contrast raster and vector formats, including typical filename extensions, colour depth, amount of transparency permitted and type of compression used.
WL40	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.
WL41	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 various file formats, creating a composite using layers and layer masks, and creating simple GIF animations using layers.
WL42	Describe the RGB, HSV and CMYK colour models.
WL43	Select colours based on the colour harmonies: monochromatic, complementary, analogous, and triadic.
WL44	Describe the five basic web page design principles (contrast, repetition, alignment, proximity, communicability).
WL45	Design a website using the five basic webpage design principles.
WL46	Critique a website using the five basic webpage design principles.
WL47	Design a small website for a mobile device taking into account the limited screen resolution, colour depth, bandwidth, and reduced keyboard.
WL48	Define the term "site organizational scheme".
WL49	Describe the characteristics of exact and ambiguous site organizational schemes, along with their sub-schemes, giving examples of where they are used appropriately.
WL50	Select an appropriate organizational scheme for a website.
WL51	Compare two websites according to their organizational scheme.
WL52	Define the term "site organizational structure".
WL53	Describe various kinds of site organizational structures.
WL54	Define the term "navigation element".
WL55	On a website, identify the primary and secondary navigation elements (e.g.,

	breadcrumb trails, site maps, and site index).
WL56	Create a website with primary and secondary navigation elements.
WL57	Use a web authoring tool (e.g., Dreamweaver) to maintain a website.
WL58	Create a website using a web authoring tool (e.g., Dreamweaver).
WL59	Define the term "search engine".
WL60	Query a search engine using AND, OR, NOT, and exact phrases.
WL61	List several search engine optimization techniques to improve a website's ranking in the search results.
WL62	Use several search engine optimization techniques to improve a website's ranking in the search results.
WL63	Analyze a website's log file to determine when visitors arrive, from where, and which pages they view.
WL64	Describe how cookies can be used in a website to customize the appearance for return visitors.
WL65	Use a tool (e.g., Flash) to create a simple two dimensional animation using multiple layers and object tweening.
WL66	Describe the fundamental characteristics and uses of e-commerce.
WL67	Describe the fundamental characteristics and uses of blogs, wikis, and RSS feeds.
WL68	Describe the fundamental characteristics and uses of content management systems.
WL69	Integrate social network sites (e.g., Twitter, Facebook, YouTube, etc.) into a website.
WL70	Integrate a blog into a website.
WL71	Integrate a wiki into a website.
WL72	Integrate an RSS feed into a website.
WL73	Use a content management system to create a website.
WL74	Create a secure e-commerce web site using an appropriate existing payment processing service.
WL75	Explain the legal issues of copyright, trademark, privacy, hate literature, libel, and jurisdiction as they apply to web content.
WL76	Given the requirements of a problem, use a scripting language to write programs to solve it.
WL77	Insert a scripting language program into a web page.
WL78	Use built-in operators, variables, and literals to create expressions in a scripting language.
WL79	Use scalar, array and hash variables in a script as necessary.
WL80	Use [the scripting language's] string manipulation features.
WL81	Describe the following constructs – selection, repetition, subprograms/functions.
WL82	Use selection structures, including if and switch.

WL83	Use repetition structures, including while, for, do.
WL84	Create a function to solve a problem.
WL85	Distinguish between void returning and value returning functions.
WL86	Use markup form tags (e.g., buttons, text, textarea, radio, checkbox, select) to collect user input.
WL87	Describe the Document Object Model (DOM) and properties and methods of form elements.
WL88	Use events and event handlers to create an interactive web page.
WL89	Describe the process of validating and submitting form data.
WL90	Save form data on the server side.
WL91	Explain the role of a CGI script in creating interactive web sites.
WL92	Compare and contrast client-side versus server-side scripting.
WL93	Write a server-side script to create a web page in response to a request, collect data from a web page visitor or send an email.
WL94	Write a client-side script to create a web page to perform actions such as form data validation.
WL95	Use server-side includes to dynamically create a web page.
WL96	Describe some of the major historical events in the evolution of the Internet and the World Wide Web.
WL97	Apply the appropriate operating system security and permissions to allow a script to execute.
WL98	Describe some of the encryption techniques used on the Internet.
WL99	Use SSL tools to create a secure connection.
WL100	Create a video or audio podcast.
WL101	Include video or audio content in a web page.
WL102	Describe the process of selecting and registering a domain name.
WL103	Describe the process of creating a valid SSL certificate.
WL104	Use authentication tools (e.g., CAPTCHA) to discourage robotic web access.
WL105	Use a validator to validate the structure of a web page.
WL106	Craft an appropriate set of keywords and description for a web page.
WL107	Craft hyperlinks and content with the search engine ranking in mind.
WL108	Define the term “Search Engine Results Page”.
WL109	Differentiate between Organic and Paid Search Engine Results Page listings
WL110	Outline a strategy for increasing the likelihood of a higher ranking on a Search Engine Results Page

Appendix 14 – Computer Science Flexible Pre-Major Agreement

The following pages provide a template institutions may use to indicate their acceptance of the Computer Science Flexible Pre-Major.

Computer Science Flexible Pre-Major Agreement

1. This Computer Science Flexible Pre-Major Agreement is intended to clarify and simplify transfer arrangements for students wishing to transfer between BC post-secondary institutions in order to take a major in Computer Science, the transfer occurring typically after the second year of study. It was developed to address challenges students experience in transferring to different institutions after second year.

2. Rather than prescribing specific courses, as is done in most other FPMs, the Computer Science FPM was created by first determining a collection of learning outcomes transferring students should have and then indicating which of an institution's courses provide those outcomes.

3. Under this agreement, sending institutions may continue to offer distinctive courses appropriate to their individual programs without restricting student access to various degree completion options. Students will find it easier to plan their programs and select their courses since the Computer Science Flexible Pre-Major subject areas are clearly identified. Students' possibilities for transfer will be maximized since the Computer Science Flexible Pre-Major is accepted by a number of participating institutions.

4. Students are advised that the Computer Science Flexible Pre-Major does not guarantee acceptance into Computer Science major programs, as acceptance depends upon students meeting both the entrance requirements of the receiving institution and any program-specific requirements specified by the receiving institution.

5. The Computer Science Flexible Pre-Major does not excuse students from non-discipline-specific requirements of programs at the receiving institution, such as English, humanities, discrete mathematics, or science credits. These must still be met prior to graduation with a Computer Science major, and students are strongly encouraged to examine the total program requirements of receiving institutions prior to applying for transfer.

6. None of the courses constituting the requirements for the Computer Science Flexible Pre-Major may substitute for upper-level requirements at the receiving institution.

7. A student who completes the basket of courses described in the Computer Science Flexible Pre-Major is deemed to have met the first- and second-year core computer science requirements of the receiving institution's Computer Science Major.

8. This Computer Science Flexible Pre-Major agreement supplements and does not supersede existing processes for establishing transfer credits, and indeed, other non-program courses will be assessed on a course-by-course basis in accordance with the BC Transfer Guide.

Institution: ____________________

Institution representative: ____________________

Name: ____________________

Title: ____________________

Email: ____________________

Signature: ____________________

Date: ____________________

The Computer Science Flexible Pre-Major requires that students take:

- Based on the learning outcomes identified as important for transferring students, at least one course in each of the required areas: algorithms and data structures, computer architecture, introductory programming, and software engineering.
- In total, a Computer Science Flexible Pre-Major consists of four or more three-credit (or the equivalent) courses. While not part of the FPM, students should also take a discrete mathematics course.

Appendix 15 – Computer Information Systems Flexible Pre-Major Agreement

The following pages provide a template institutions may use to indicate their acceptance of the Computer Information Systems Flexible Pre-Major.

Computer Information Systems Flexible Pre-Major Agreement

1. This Computer Information Systems Flexible Pre-Major Agreement is intended to clarify and simplify transfer arrangements for students wishing to transfer between BC post-secondary institutions in order to take a major in Computer Information Systems, the transfer occurring typically after the second year of study. It was developed to address challenges students experience in transferring to different institutions after second year.

2. Rather than prescribing specific courses, as is done in most other FPMs, the Computer Information Systems FPM was created by first determining a collection of learning outcomes transferring students should have and then indicating which of an institution's courses provide those outcomes.

3. Under this agreement, sending institutions may continue to offer distinctive courses appropriate to their individual programs without restricting student access to various degree completion options. Students will find it easier to plan their programs and select their courses since the Computer Information Systems Flexible Pre-Major subject areas are clearly identified. Students' possibilities for transfer will be maximized since the Computer Information Systems Flexible Pre-Major is accepted by a number of participating institutions.

4. Students are advised that the Computer Information Systems Flexible Pre-Major does not guarantee acceptance into Computer Information Systems major programs, as acceptance depends upon students meeting both the entrance requirements of the receiving institution and any program-specific requirements specified by the receiving institution.

5. The Computer Information Systems Flexible Pre-Major does not excuse students from non-discipline-specific requirements of programs at the receiving institution, such as English, humanities, discrete mathematics, or Information Systems credits. These must still be met prior to graduation with a Computer Information Systems major, and students are strongly encouraged to examine the total program requirements of receiving institutions prior to applying for transfer.

6. None of the courses constituting the requirements for the Computer Information Systems Flexible Pre-Major may substitute for upper-level requirements at the receiving institution.

7. A student who completes the basket of courses described in the Computer Information Systems Flexible Pre-Major is deemed to have met the first- and second-year core computer science requirements of the receiving institution's Computer Information Systems Major.

8. This Computer Information Systems Flexible Pre-Major agreement supplements and does not supersede existing processes for establishing transfer credits, and indeed, other non-program courses will be assessed on a course-by-course basis in accordance with the BC Transfer Guide.

Institution: ______________________________

Institution representative: ______________________________

Name: ______________________________

Title: ______________________________

Email: ______________________________

Signature: ______________________________

Date: ______________________________

The Computer Information Systems Flexible Pre-Major requires that students take:

- Based on the learning outcomes identified as important for transferring students, at least one course in each of the required areas: algorithms and data structures, hardware, information management (database), introductory programming, networking, software engineering, and web learning.
- In total, a Computer Information Systems Flexible Pre-Major consists of seven or more three-credit (or the equivalent) courses.