TO : Prof. C.L. Philip Chen, Dean of FST
FROM : K.H. Vat, Senior Instructor, DCIS – FST
DATE : December 06, 2010
SUBJECT : Concerns and Responses over Course Evaluation Results
CC : Professor GONG Zhiguo, Head of Department of CIS

I am writing with my reflective responses to the result of student course evaluation on *SFTW241 Programming Language Architecture (I)* (Spring-2010 Semester), which has been passed to me by Prof. GONG Zhiguo, Head of Department of CIS earlier last month. Indeed, this course has been under my teaching assignment since 1993. Please be assured that I share your concerns with the result, which has aroused my attention to look into my course portfolio for this Spring-2010 semester, including the whole course design with all possible evidence of learning produced by students, the lecture and tutorial materials rendered as well as the process of teacher and student peer assessment based on the course enactment details. The whole course archive has been studied based on the version kept track of through our UMMoodle (e-Learning system) environment, including all the course materials recorded up 2010JUN30 (end of the Spring-2010 semester).

Course Design of SFTW241-2010
This is the first of a 2-course sequence (SFTW241 compulsory + SFTW342 optional) introducing the concepts, techniques, and models of computer programming. The concepts are organized in terms of computation models introducing different techniques for programming and reasoning about programs. Example computational models covered in this course include the imperative and object-oriented programming and reasoning techniques. Each computation model is based on a core language introduced in a progressive way, by adding concepts one after the other. The languages explored throughout the course include: ANSI C, ANSI C++, and Java.

Contextually, the knowledge areas and topics, as well as intended learning outcomes (ILOs) selected for exploration in the Spring-2010 semester include the following:

Concepts	**Topics**	**ILOs**
Overview of Programming Languages	History of programming languages; brief survey of programming paradigms (procedural, object-oriented and concurrent languages); the effects of scale on programming methodology	1. Describe the evolution of programming languages illustrating how this history has led to the paradigms available today. 2. Identify at least one distinguishing characteristic for each of the programming paradigms covered in the course. 3. Evaluate the tradeoffs between the different paradigms, such as structured programming and object-oriented programming, considering such issues as space efficiency, time efficiency, safety and power of expression. 4. Distinguish between programming-in-the-small and programming-in-the-large.
Virtual Machines	Concept of a virtual machine; hierarchy of virtual machines; intermediate languages	1. Describe the importance and power of abstraction in the context of virtual machines. 2. Explain the benefits of intermediate languages in the compilation process.
Introduction to Language	Comparison of interpreters and	1. Compare and contrast compiled and

Translation	compilers; language translation phases (lexical analysis, parsing, code generation, optimization)	interpreted execution models, outlining the relative merits of each. 2. Describe the phases of program translation from source code to executable code and the files produced by these phases.
Declarations and Types	The conception of types as a set of values together with a set of operations; declaration models (binding, visibility, scope and lifetime); type-checking overview; garbage collection	1. Explain the value of declaration models, especially with respect to programming-in-the-large. 2. Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size. 3. Demonstrate different forms of binding, visibility, scoping, and lifetime management. 4. Defend the importance of types and type-checking in providing abstraction and safety. 5. Evaluate tradeoffs in lifetime management, such as reference counting versus garbage collection.
Abstraction Mechanisms	Procedures, functions, and iterations as abstraction mechanisms; parameterization mechanisms (reference versus value); activation records and storage management; type parameters and parameterized types; modules in programming languages	1. Explain how abstraction mechanisms support the creation of reusable software components. 2. Demonstrate the difference between call-by-value and call-by-reference parameter passing. 3. Defend the importance of abstractions, especially with respect to programming-in-the-large. 4. Describe how the computer system uses activation records to manage program modules and their data.
Object-Oriented Programming	Object-oriented design; encapsulation and information-hiding; separation of behavior and implementation; classes and subclasses; inheritance (overriding, dynamic dispatch); polymorphism (subtype polymorphism versus inheritance); class hierarchies; collection classes and iteration protocols; internal representations of objects and method tables	1. Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance, and polymorphism. 2. Design, implement, test, and debug programs in an object-oriented programming language. 3. Describe how the class mechanism supports encapsulation and information hiding. 4. Design, implement, and test the implementation of "is-a" relationships among objects using a class hierarchy and inheritance. 5. Compare and contrast the notions of overlapping and overriding methods in an object-oriented language. 6. Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
Type Systems	Data types as set of values with set of operations; data types including elementary types, product and co-product types, algebraic types, recursive types, arrow (function) types, and parameterized types; type-checking models; semantic models of user-defined	1. Formalize the notion of typing. 2. Describe each of the elementary data types 3. Explain the concept of an abstract data type. 4. Recognize the importance of typing for abstraction and safety.

	types, including type abbreviations, abstract data types, and type equality	5. Differentiate between static and dynamic typing. 6. Differentiate between type declarations and type inference. 7. Evaluate languages with regard to typing.
Programming Languages Design	General principles of language design; design goals; typing regimes; data structure models; control structure models; abstraction mechanisms	1. Evaluate the impact of different typing regimes on language design, language usage, and the translation process. 2. Explain the role of different abstraction mechanisms in the creation of user-defined facilities.

It is also expected that at the end of the study of the Spring-2010 semester of SFTW241, the following university-wide ILOs are to be accomplished, too:

- Find, evaluate and use appropriate learning resources
- Work cooperatively in teams and small groups
- Demonstrate versatile and effective communication skills, both verbal, written, and hands-on
- Use content knowledge and intellectual skills acquired in the course to become continual learners

Pedagogical Practice
Pedagogy: a mixture of didactic teaching and problem-based learning (PBL) approach where students are trained to work in professional groups (first in pairs, then in groups of pairs) to do collaborative project work. It is my practice to explain the syllabus and the pedagogy of my practice to each class of my teaching assignment, at the beginning of each semester. This was indeed what I repeatedly did for SFTW241 through our UMMoodle environment in the spring of 2010. The work of the whole semester, based on the records of student-groups' reports, actually reflect clearly the track we have been following according to the syllabus stated. Please refer to the UMMoodle course site of SFTW241-2010 for specific student coursework completed for more details.

Ever since 2000, the pedagogy of problem-based learning (PBL) has been adopted in SFTW241 (along with the traditional didactic teaching), where the class of students is transformed, at the semester's start, into groups of learning units, each comprising a suitable number of students depending on the class enrollment. There were 8 pairs from Class A and 13 pairs from Class B in the spring-2010 semester. Each pair has two or at most three members (to round off the odd number of students in class) to participate in pair-programming exercises throughout the semester. The two roles in each pair are the driver and the navigator who need to learn programming together through a timely turn of roles, instead of a lone programming individual. Indeed, the 21 pairs of students were also organized into 12 teams of student groups to proceed with group project work throughout the semester, to participate in collaborative project-based learning. This design is important to develop students' ability to "learn to learn" with companion, especially in the language-learning scenarios of group-based project work, namely, understanding, designing, implementing programmatic solutions, and evaluating the design differences across programming languages to match (express) the given solution, especially in the context of structured and object-oriented programming solutions, using C++ and Java. The essential pre-requisite knowledge of at least one procedural programming language, and basic practice of doing programming assignment is required, such as applying ANSI C acquired in such courses as SFTW120 and SFTW111, to implementing basic data structures and algorithms.

Coursework Design
The design of SFTW241 coursework in the Spring-2010 semester essentially comes down to the following: six lab exercises, one per week; one assignment in April-2010; one project in May-2010.

Laboratory Exercises

Lab Session	Context	Topics in Programming Practice
Lab #1	Review in basic ANSI C programming in problem solving	Pair-Formation Scenario
Lab #2	Peer assess Lab #1 through analysis and reflection of individual work presented based on specific assessment rubrics	Program Interpretation of problem solving in ANSI C
Lab #3	Measure individual proficiency in C problem solving based on a self-proposed problem given some consistent assessment rubrics	Demonstration of various C-based programming constructs to illustrate personal proficiency in programming practice, bearing in mind the rubric items for evaluation
Lab #4	Enhance individual proficiency in programmatic problem solving in the context of C-based structured programming technique	Build a simple virtual machine, from a concise description of hardware requirements, such as word length, addressing modes, and instruction sets, including software interfaces, and input/output formats.
Lab #5	Peer assess Lab #4 by applying reusable and consistent assessment rubrics, by analyzing the problem solving steps and by test-running the completed program(s) submitted, as well as by comparing to some previous work from students several semesters earlier.	Analyze programs and interpret whether or not the solution works in the specific problem context for Lab #4, by dissecting the programmable solution (program design) in terms of different artifacts as suggested in the rubrics.
Lab #6	Solve a peer review allocation scenario by creating a programmable solution, to be implemented first in C and then in C++, comparing the methodical thinking in structured and object-oriented programming	The problem scenario requires help in allocating to a class of students a review exercise: Each student is to review up to two homework exercises from two other students in the same class, not including his or hers; Each piece of homework is to be reviewed by at most two students.

Team Assignment in April-2010
This is a team-based assignment in programming respectively with ANSI C, and Java, concerning the implementation of the lexing and the parsing process for a C-like language called CLite. Each team composed of two pairs of students, is obliged to experience the differences in methodical thinking between structured programming (in C) and object-oriented programming (in Java) in solving the programmable problems in the partial realization of CLite and the following general requirements describe what our students are expected to fulfill:

- Each team is required to come up with a PLAN to define the work of each of the two pairs, and of each member in the pair. Namely, students must have a clear division of work (with full description of time commitment from each team member and pair member) based on which the assessment of each student's accomplished work is to be performed. The plan must be visible in the Forum TeamSpace.

- Each team is to follow the documentation rules as suggested in class to put the identity of each team, of each pair within the team, and of each individual team member in the different artifacts of work to be submitted.
- This is not just a programming assignment, but a discussion and illustration assignment that each student need to indicate his or her ongoing work through the TeamSpace, the PairSpace, as well as the Personal Space (the personal wiki created earlier). The TeamSpace and the PairSpace are the affiliated forums for student convenience just as those with the lab exercises students finished earlier.
- The topics of interest deemed appropriate in this assignment include the following: a) problem statement, b) situational analysis, c) method of solution, d) data structures and algorithms created, e) program design, f) program interface with users, g) automated or informated operations, h) Java and ANSI C implementation of the program, i) test-run with sensible data in Eclipse JDT and Eclipse CDT under Window XP, and j) program documentation.

Group Project in May-2010
This is a month-long (four phases in four weeks) group project in programmatic problem solving in the context of programming languages design, using the problem-based learning paradigm. Namely, students need to work in small groups, with specific group roles, to accomplish the various tasks of the projects:

- Peer review of the work in Team Assignment
- Redo of the Team Assignment using an object-oriented approach, but the program must be written in ANSI C++ instead of C or Java.
- Write a simple hardware emulator and the assembler for a simple machine language, called Simpletron Machine Language (SML), pulling together almost all the learning accrued through the previous exercises in programming. To facilitate student learning, a case solution from students' work from several years ago, has been made available. The case solution uses C++ as a language, but students in SFTW241-2010, must use Java as the language instead, encouraging them to learn as much as possible from the C++ solution, with critical analysis of the program design and language constructs used.
- As a team project, this proves quite a challenge for our PBL (problem-based learning) students. They are compelled to learn a lot to solve the problem.

Final Examination in June-2010
The final examination of SFTW241 is based on the project work in May, especially for the internal evaluation of individual team members in the project work. Students are invited to provide a separate confidential evaluation, one for each team members, and write down their qualitative comments based on the evaluation questions rendered. The overall course layout reflects an underlying design to introduce to students the importance of peer assessment of one another's work through pinpointing the differences between structured programming and object-oriented programming, in the context of using C, C++ and Java programming to explore the language design issues in computer programming.

Rubrics for Student Self and Peer Assessment in SFTW241-2010
It is important to mention the framework of assessment (rubrics) suggested for students to do self- and peer assessment of their work throughout the semester as follows:

ANSI C: General Concerns (pre-requisite for review)
1. *Structured Techniques for Computing*
   Any specific diagrams to explain the program structure (or design) performed in completing the program assignment.
2. *Modular Programming and Control of Complexity*
   Any specific use of the technique 'divide and conquer', and 'program modules of utility procedures and/or functions to solve the problem.

3. *Programming by Stepwise Refinement*
   Any specific demo to express different levels of abstraction to refine programming solutions to the problem.
4. *Program Documentation*
   Any specific program documentation (in-program or design-based) used to ease the job of program modification.
5. *Post-Programming Report*
   Any specific use of documentation, say, "typescript" in UNIX and any effort of beautification using MS-Word to present a report good to read.
6. *Problem Formulation*
   Any explanation of the problem context based on which the programming solution is attempted.
7. *Algorithmic Exposition*
   Any clear explanation of the method used to handle the programming problem according to the specific policies; say, each pair is to evaluate at least three pairs; and each pair is to receive score from only three pairs.
8. *Programming Style*
   Any use of legible text-typing of the program following the strategy of not allowing any wrap-around (per horizontal line) in the source coding.
7. *Separate Compilation*
   Any use of the technique to break down a large (single) program file into several logically coherent modules or component units, including the header files and the corresponding implementation files, as well as the test driver (main program).
8. *Efficacy*
   Any clear indication that the program works as expected, in view of the test data.

C++: Add Specific Concerns (developed during the semester)

1. *Encapsulation*
   Any use of the technique "separation of concerns" through specifying both the interface (what) and implementation (how) of a specific object (advanced form of abstract data type), say, using the "class" construct in C++.
2. *Inheritance*
   Any use of the inheritance technique to abstract commonality among different classes of objects. Any specific specification of the IS-A relationship in the design of various objects.
3. *Polymorphism*
   Any use of the polymorphism technique to define using the same name, the virtual functions, respectively performing different task context, say, a circle, and a cylinder objects each have the "paint" function, which paints different shape according to the contextual object, i.e., a circle or a cylinder. The use of the keyword "virtual" in C++ is expected.
4. *Structured-vs-Object-Oriented Programming*
   Any indication of program design using structured programming (SP), or object-oriented programming (OOP) technique. How SP or OOP is the program completed? Any combined use of SP and OOP expressed in the program?
5. *Lessons Learned*
   Any specific lesson learned indicated in the brief report, detailing the learning acquired after applying the C++ programming language to solve the problems at hand.

Java: Add Documentation and Design Concerns (*developed during the semester***)**

1. *User-Interface* (UI)
   Any specific arrangement in the source listing of Java, indicating the UI area containing the front-end user-interface components.
2. *Application Logic* (AL)
   Any specific arrangement in the program source listing of Java, indicating the AL area containing the mid-layer program units, such as the use of imported objects, routines, or user-defined data types.

3. *Data Resources* (DR)
   Any specific arrangement in the program source listing of Java, indicating the DR area containing the back-end data such as external data files.

It is the position of SFTW241 that programming proficiency in at least one high-level programming language is a pre-requisite so as to transfer the knowledge from one language to another even across two different programming paradigms, such as structured (imperative) programming and object-oriented programming (from ANSI C to C++ or Java). So, developing programming proficiency in C++ or Java is not the core learning goal in SFTW241, but the ability to appreciate the intricacy in the design of different programming languages, through creating programmable solutions in various procedural and object-oriented programming languages. Namely, these include the intended learning outcomes as delineated earlier for SFTW241.

Design of Student Learning Experience

In designing student learning experience in SFTW241-2010, I have born in mind several pedagogical objectives. Namely, how could my course design achieve the following in student learning?

- Developing student responsibility in active learning
- Making learning meaningful to their future study or vocational goals
- Promoting overt knowledge construction with down-to-earth hands-on experience
- Performing learner assessments to stimulate further learning
- Showcasing learner achievements in terms of accessible records

In other words, I must install a method of teaching which could facilitate student learning to come close to the above-mentioned objectives. My past experience in constructivist design of student learning has enabled my recognizing the potential of problem-based learning (PBL), whose effective use has rendered many a flexibility and possibility in producing students' initiative in their own learning under different course scenarios.

What is entailed in PBL?

The notion of PBL is based on the premise that students learn more effectively when they are presented with a problem to solve rather than just being given instruction. Pedagogically, students have to identify and search for the knowledge they need to approach the problem. When applied to student learning activities, PBL could be decomposed into several stages of participation, which help develop in students, self-directed learning and problem-solving skills while they interact, discuss, and share relevant knowledge and experience:

- *Problem analysis stage*: Students, divided into small groups and assigned a facilitator (played by the teacher), are respectively presented a problem scenario without much instruction given. They generate ideas about possible solutions to the problem based on what they already know. They then define what they need to know by identifying the key learning issues and formulate an action plan to tackle the problem.
- *Information gathering stage*: A period of self-directed learning follows. Students are responsible for searching for relevant information. They are largely engaged in just-in-time learning as they are seeking for information when their need to know is greatest.
- *Synthesis stage*: After a specified period of time, students reconvene and reassess the problem based on their newly acquired knowledge. They become their own experts to teach one another in the group; they use their learning to re-examine the problem. In the process, they are constructing knowledge by anchoring their new findings on their existing knowledge base.
- *Abstraction stage*: Once the students feel that the problem task has been

successfully completed, they discuss the problem in relation to similar and dissimilar problems in order to form generalizations.
- *Reflection stage*: At this stage, students review their problem-solving process through conducting a self- and/or peer-evaluation. This stage is meant to help students' meta-cognitive ability as they discuss the process and reflect on their newly acquired knowledge.

The design of a PBL course (or even curriculum) addresses directly many of the recommended and desirable outcomes of a quality undergraduate education; specifically, the ability to do the following as recommended by the Boyer's Report:

- Think critically and be able to analyze and solve complex, real-world problems
- Find, evaluate, and use appropriate learning resources
- Work cooperatively in teams and small groups
- Demonstrate versatile and effective communication skills, both verbal and written
- Use content knowledge and intellectual skills acquired at the university to become continual learners

Thereby, PBL is designed to actively engage students, divided in groups, in opportunities for knowledge seeking, for problem solving, and for the collaborating necessary for effective learning. At the heart of PBL are some real-world problems (or scenarios) used to motivate students to identify and research the issues and principles they need to know to work through those problems. In the context of SFTW241-2010, the overall course design is consistent with this design philosophy. But, it is also a very demanding experience for our students.

Perceived Student Difficulties in Learning

Obviously, it is my observation, in almost every spring semester of SFTW241, that our students are largely under-prepared to get involved actively in their own study, owing to either their lack of solid motivation and ability in terms of "managing to learn", or their shabby foundations in content knowledge to do programming proficiently, as second-semester sophomore students. The former has something to do with our lack of carefully crafted First-Year Experience Program (FYE) throughout our Faculty (such as lack of such freshman course like *Professional Skills*), while the latter with our yet-to-be-harmonized design of CIS curriculum in programming fundamentals (even our new curriculum in Computer Science is still very weak). More importantly, their soft skills, say critical thinking and collaborative analysis in teamwork is very weak, with almost zero experience/education in formal group work, like the IEEE suggested affinity research group model (something that should be included in the *Professional Skills* course). Students' typical mentality about college teaching and learning still stays with the didactic model as in the secondary school: namely, their only understanding of teaching is the knowledge transmission (KT) model. They enter the classrooms to learn by listening, and if the teacher is not talking (or presenting) as expected, he or she is considered not teaching. My students of SFTW241, as far as I could perceive, semester by semester, largely are not aware of the learning facilitation (LF) model of teaching, based on which our UM's current emphasis in outcomes-based approach to student learning (an essence of our elite undergraduate education) is founded.

Outcomes-Based Teaching and Learning (OBTL) – My Reflections

I personally singled out specific lectures at the beginning of each semester to explain to and to remind my students continually throughout the semester of what is expected of them under our outcomes-based approach – a meaningful and deep approach to learn, in terms of the following: Namely, UM is convinced that an OBE (outcomes-based education) approach to student learning is a learner-centered approach to curriculum and course design, and to teaching and learning, that is focused upon what the students are expected to learn and to do, rather than what the

teachers expects to teach and to do. This OBE approach is mainly powered up through the constructive alignment of three important elements in action: ILO's (intended learning outcomes), TLA's (teaching and learning activities), and AT's (assessment tasks), including the provision of assessment rubrics:

- *Intended Learning Outcomes* (ILO's) are what students are expected to be able to do at the end of a lecture, a course, a project, a field trip or a program of study. They are expressed from the student perspective, in the form of some action verbs (identifying the learning outcomes), and related to criteria for assessing student performance. They are referred to as ILO's because in good learning environments, students may also learn many additional things about the academic subject, working with others, dealing with difficult people, teamwork, and other living and learning skills such as adaptability with emerging Web technologies and social media, which are not necessarily included in the ILO's.
- *Teaching and Learning Activities* (TLA's) are activities designed by academic staff (professors and instructors) to help students achieve the learning outcomes of the course, of the tutorials, of the lab sessions, of the lectures, of the projects, or of the field trips. The TLA's must be explicitly related to each ILO. For example, if an ILO is that students will develop the ability to solve particular types of problems, lecturing students about how to solve such problems will not be sufficient. Students will need practice, support and feedback in solving such problems.
- *Assessment Tasks* (AT's) are procedures designed to assess the related ILO's after the specific TLA's are identified that will help students achieve the ILO's. Oftentimes, creating the appropriate AT's is an iterative process involving different levels of review, revision, and development. For example, if an ILO is that students will develop skills in oral communication, then asking student to write an essay about oral communication does not assess the related ILO. Students need to engage in an act of oral communication which is assessed accordingly. Thereby, AT's could come in various forms such as essay-type assignments, projects, presentations, quizzes, role-plays, e-portfolio collection, and many others, our teachers ask students to do to demonstrate evidence that a particular ILO has been achieved.
- *Assessment Rubrics* (AR's) are standards (or criteria) explicitly devised to measure the performance of student achievement in the context of ILO's. They must be developed after the AT's have been identified. For example, a course of study might define an 'A' as showing evidence of original thought or being able to critically analyze evidence, but a 'D' as being able to reproduce what was taught with no evidence of critical analysis or original thought. Each grade needs to have a grade descriptor, describing explicit differences between the grades. And grades, as a form of criterion-referenced assessment, are meant to describe what students can or cannot do rather than how their performance compares to other students.

At the University of Macau (UM), we consider the outcomes-based approach to student learning as an expression of UM's commitment in elite undergraduate education (http://www.umac.mo/curriculum_reform/), taking into account the holistic concerns of student development. This outcomes-based education (OBE) approach calls for the articulation of what we expect our students to learn and to become, and the collection of evidence to determine whether our students have acquired the learning expected. It is believed that clear understanding and articulation of intended learning outcomes (ILOs) should facilitate the design of an effective curriculum and appropriate assessments to measure student achievement, as well as to provide strategic planning of personalized learning processes for individual students. Yet, this approach implies (indeed, demands) active participation from students (not just teachers) in the content and process of the conversational practice and knowledge construction in class. Both students and teachers must take joint responsibility for learning. As elaborated many times in class, that PBL is a type of collaborative learning with the following characteristics, has somehow not been effectively received by many of our SFTW241 students:

Student responsibility involves:

- Preparing for lectures by doing the reading indicated for each lecture;
- Participating in discussions during the meeting time, and virtually during our online synchronous and asynchronous modes of forum discussions;
- Active involvement in journaling your learning, asking questions and finding answers;
- Being courageous and speaking your mind

Teacher responsibility as facilitator involves:

- Designing and guiding the collaborative learning process (see Design of Student Learning Experience);
- Facilitating in-class conversational practices;
- Steering our course of learning;
- Providing inputs and feedbacks where necessary

Frankly speaking, I am yet to hear conscientious sharing at the departmental level from colleagues who have experienced their teaching and learning with students, from this level of depth over my almost 18 years of service at the University of Macau. Perhaps, it is time for us to share with one another our experience with such an initiative as **Peer Review of Teaching**, starting from FST or FST-DCIS, through our course portfolios to be incrementally built up for program accreditation purpose. It is really a good time to seriously think about that. I firmly believe that there is a lot we can learn from one another, such as those whose student rating always stay close to the maximum. Their ways to inspire students together with the materials they prepare and present in class must represent excellent lessons for others to model after in the Faculty. We could actually hold on a regular basis, say, semester-end retreat at a department or faculty level, such activity as **Teaching & Learning Experience Sharing Retreat** (1 or 2 days being good enough). Meanwhile, it is also suggested to make visible the **course portfolios** of all teachers from the faculty so that we could learn from one another by gaining access to such important teaching and learning intellectual assets, so much valued by the University that they are the foundations of the recognized **Scholarships of Teaching and Learning** (SoTL) from UM. In view of the modern educational technology, we could also install an *electronic course portfolio system* to facilitate this important milestone from FST. I may be dreaming, but this is not a distant dream given the mandate of elite undergraduate education from the Rector that UM is to become a world-class university.

Course Complaints from Students
According to my conversations with some of the students after the final examination, the following findings seem to have surfaced among groups of students:

1. Many students feel that there are too many coursework assigned or learning activities developed during the semester. They face pressure to handle the course workload, even on the pair/team-programming basis.
2. Some students failed to go deep enough into both C++ and Java within the two months' duration allowed in a semester, to acquire the necessary training to get the most out of the assignments. Students hope that there could be a slow-down of the course progress, and a reduction of coursework.
3. There are students who are realistic enough to see instructor's grade on their individual assignment instead of the formative comments along the way or score assigned only at the end of all the assignments. They need more instructor quantitative feedback for their work in terms of score, instead of qualitative feedback for how to improve their work. This is obviously a signal of the conventional didactic tradition of teaching and learning in high schools.

Admittedly, ever since I adopted the PBL approach of course delivery along the didactic mode of teaching in SFTW241, I have indeed observed students' difficulty to get used to the new style of learning, which requires their initiative to learn to manage their time resources really well. Yet, with the pair/team-programming experience of this past semester, it is my realization that many of the students have difficulties of performing C programming activities under Windows using IDEs like Eclipse CDT to create programs, to compile using the linked C compiler (through say, cygwin) under CDT, debugging the program iteratively and capturing the test run using Eclipse CDT console. These basic programming activities sound strange to them. This simple observation, though seemingly fixed after the beginning hands-on tutorials, might have nonetheless persisted to snowball into their semester complaint of heavy workload in the course of programming exercises respectively in C, C++, and Java. Assuredly, this difficulty must be overcome with an open heart from both teachers and students as my experience shows.

Course Context for Improvement

The context of SFTW241 *Programming Language Architecture (I)* is inherently not purely about how to program, but about the concepts and paradigms of programming languages, which are devices necessary for a critical evaluation of existing and future programming languages. Namely, we need to discuss the design issues of the various language constructs, examine the design choices for these constructs in some of the most common languages, and critically compare design alternatives. The ability to program in two or more different language paradigms (structured with ANSI C and object-oriented with C++ or Java) is preferably established before students embark on this course. Unfortunately, in our program, we have not designed the course sequence such that this preferred pre-requisite is enforced. Consequently, the course carries the responsibility to equip students with programming knowledge in a "to-be-discussed" language such as C++ or Java, before the various programming constructs belonging to that language could be elaborated. This is like putting several courses' materials in one. There are enough materials in learning C++ or Java programming, to fill up a year's of study. But, typically, we have less than one month to cover the minimum set of materials in each language to get going in SFTW241. Unfortunately, we have yet a sequence of courses dedicated to teaching C++, or Java under the cover of *Programming Methodology (I) or (II)* throughout the four-year program of CIS, let alone in the semesters before SFTW241 being taken by students. In the current course design of SFTW241, I have consulted the advice from Computing Curricula 2001 – Computer Science (Final Report by ACM and IEEE Joint Task Force on Computing Curricula, dated December 15, 2001; pp. 113-117) to select the set of core topics with the indicated ILOs (see earlier in the section of Course Design) for inclusion into the course content delivered through our selected text and course e-resources.

Suggestions for Improvement in the Coming Semester

Sincerely, I may not be able to answer to every student his or her individual complaint about my course delivery. Still, as a teacher sincere enough to offer my suggestions, here are some adjustments ready to be introduced in the coming semester:

1. Keep the original focus on the differences between structured programming and object-oriented programming, but the languages to be discussed are confined to C++ with ANSI C. That way, we could devote more time (twice the current amount) to the study of C++, hoping to go deeply enough to help students acquire the best they can afford. And we concentrate on comparing the programming language constructs across two major paradigms: imperative and object-oriented. Given the richness of the C++ language, in terms of features supporting object-oriented programming, there is enough to cover in the remaining semester. However, the tradeoff is to postpone the study of Java to SFTW342, the continuation of SFTW241, but an elective course offered to junior students, covering the specific aspects of language syntax and semantics, including the comparison among concurrent paradigm, object-oriented

paradigm and imperative paradigm of programming.

2. Refine the current course offering to provide two new sophomore courses in *Programming Methodology*, one covering the imperative and object-oriented paradigms in C and C++, and the other covering the object-oriented and concurrent paradigms in C++ and Java. Combine the original SFTW241 course with SFTW342, and offer the combined course as a junior compulsory course renamed as "Concepts of Programming Languages" with the assumed context of studying the design issues of various programming languages including C, C++, Java, and perhaps C# (another language of emergent importance). Certainly, the pre-requisites for this course must be the successful completion of the two new courses in *Programming Methodology (I)* and *(II)*. That way, we can devote whole semesters to studying the specific language constructs in the context of language design issues.

Responding to Students' Inertia to acquire Text and Reference books

It is my experience that students are not willing to buy the textbook and any suggested references for reason of high costs in Macau, compared to the cheaper Chinese translated version of similar books inside mainland China. This presents some difficulty to match my lecture materials with those coming from students' own progress of study. I have therefore resorted to posting online the pdf copies of the various chapters from the designated text, through our UMMoodle site of SFTW241. I also declared many times in class that those who choose not to purchase the textbooks must make their necessary photocopies of the related chapters from the Library copy of the same text or references or download the pdf chapter copies from our UMMoodle site. Yet, except for specific stipulation that such chapters are required during examination, I still find students coming to class or lab without the appropriate text or references. I believe somehow there must be a way to warn students of their responsibility and consequences of not acquiring the adopted texts. And I take it as an important area of improvement in the coming semester.

Attention to the Pre-requisites for SFTW241: SFTW210 should better be included

It is suggested that instead of requiring only SFTW120 as the pre-requisite for SFTW241, SFTW210 *Data Structures and Algorithms (II)*, should better be included because that way, we can have students embarking on SFTW241 having had at least two courses (SFTW111 and SFTW210) of programming in ANSI C before coming to learn C++ and/or Java as a means to appreciate the design issues of programming languages in SFTW241. It is my survey findings obtained at the beginning of the spring semester that many students actually failed the two courses SFTW111, and SFTW210 while embarking on SFTW241. This could be a big problem in facilitating our lectures and coursework.

Some Words of Assurance

Pursuing excellence in teaching and research has been the essential endeavor of my professional practice as a teacher-researcher. One of my personal guideposts in life is: Trying out ideas in practice as a means of improvement and as a channel of knowledge synthesis through doing something about it. Putting it in my teaching context, I believe what I expect from my students in the spring-2010 semester of SFTW241 is their minimum amount of willingness to learn to learn and to reflect on what they do, and then answer to themselves the simple question of how much they have learned to understand the concepts of programming languages, given their semester's work with C, C++ and Java respectively doing imperative programming and object-oriented programming exercises. In the short term, I expect my students to solve the problem successfully and understand the content behind the programming solution. In the long term, I expect them to develop their ability to appreciate the process of language design, covering such abilities as:

- To reason clearly about programming languages through evaluating the wisdom

and utility of the decisions made in the process of language design.
- To break down a language into its major components, and each into smaller pieces so as to focus on competing alternatives.
- To become aware of a consistent and general set of terms for the components out of which programming languages are built, and the concepts on which they are based.
- To see below the surface appearance of a language to its actual structure and descriptive power, through understanding that many language features which commonly occur together, are in fact, independent and separable.

My Inner Sharing as a Teacher

Effective classroom teaching requires professional commitment in the act of teaching, which is founded on the planning and implementing of instructional activities and the assessment of student performance. PBL-based OBE represents an important commitment I have made to improve students' learning while they are preparing their journeys into professional software practitioners or computer scientists in the academic arena. In the context of course evaluation by students, to those who say that we need weights and measure in order to enforce accountability in education, my response is, yes, of course we do, but only under certain conditions that are not being met today. Namely, we need to make sure (1) that we measure things worth measuring in the context of authentic education; (2) that we know how to measure what we set out to measure; and (3) that we attach no more importance to measurable things than we attach to things equally or more important that elude our instruments.

As Parker, J. Palmer in his 1998 book, *The Courage to Teach*, says, "Good education is always more process than product. If a student has received no more than a packet of information at the end of an educational transaction, that student has been duped. Good education teaches students to become producers of knowledge and discerning consumers of what other people claim to know. Yet, good education may leave students deeply dissatisfied at least for a while. I do not mean the dissatisfaction that comes from teachers who are inaudible, incoherent, or incompetent. But students who have been well served by good teachers may walk away angry – angry that their prejudices have been challenged and their sense of self shaken. That sort of dissatisfaction may be a sign that real education has happened. It can take many years for a student to feel grateful to a teacher who introduces a dissatisfying truth. A marketing model of educational community, however apt its ethic of accountability, serves the cause poorly when it assumes that the customer is always right." (p.94)

Now that the present workload situation allows many a student to take a minimum of six to a maximum of eight/nine courses per semester, it is easy for students to complain of one instructor's coursework taking up too much the student's personal study time that should be devoted to another instructor's assignment, under the pretext of not being able to afford "too much time" in only one course, to ignore the work required by another course. In light of that we might need some guideline to regulate the workload to be assigned in each course of study. That obviously puts constraints on any instructor to design a suitable course delivery plan. What about limiting the number of courses a given student could take? Given the current teacher-centered approach, emphasizing the many course hours per week, of knowledge transmission to students, it might not be smooth to shift the focus from what we are supposed to teach (students sit and listen), to what is expected of students to learn (students mobilized to acquire learning by doing). That is indeed another point of balance between the instructivist mode of teaching and the constructivist approach to learning.

We cannot afford the commercialization of higher education to use undergraduate student course evaluation as a holistic measure to cheapen our teaching efforts. We do not teach to increase our student rating. We must provide the substance, say, concrete evidence of quality teaching through the provision, say, of course portfolio system. The term *student-centered* must not become an oxymoron; so, I prefer the term

learner-centered instead, although both terms imply a focus on student needs. Yet, being student-centered carries the connotation that might give rise to the idea of education as a product, with the student as the customer and the role of the faculty as one of serving and satisfying the customer. Faculty members resist the student-as-customer metaphor for some very good reasons. When the product is education, especially in a four-year undergraduate degree program, the customer cannot always be right; there is no money-back guarantee, and tuition dollars do not "buy" the desired grades. Nonetheless, the learner-centered approach orients to the idea of "product quality" constructively. Being learner-centered is not about cowering in the competitive academic marketplace. It is not about kowtowing to student demands for easy options and is not about an ethically irresponsible diminution of academic standards in an attempt to placate "shoppers" who may opt to purchase educational products elsewhere. It is about creating climates in classes and on campus that advance learning outcomes. It is an orientation that advocates for more, not less learning. It is about offering a quality learning experience through our elite undergraduate education program.

More relevantly, the learner-centered context focuses attention on learning: what the student is learning, how the student is learning, conditions under which the student is learning, whether the student is retaining and applying the learning, and how current learning positions the student for future learning. The student is an important part of the equation. Indeed, we make the distinction between student-centered instruction and teacher-centered instruction as a way of indicating that the spotlight has moved from teacher teaching to student learning. When instruction is student-centered, the action focuses on what students (not teachers) are doing. Since the instructional action now features students, the student-centered orientation accepts, cultivates, and builds on the ultimate responsibility students have for learning. Teachers cannot do it for students. They may set the stage, so to speak, and help out during rehearsals, but then it is up to students to perform, and when they do learn, it is the student, not the teacher, who should receive accolades. This is the essence in a quality undergraduate learning experience. How much have we taken this into account when we design the teaching evaluation system?

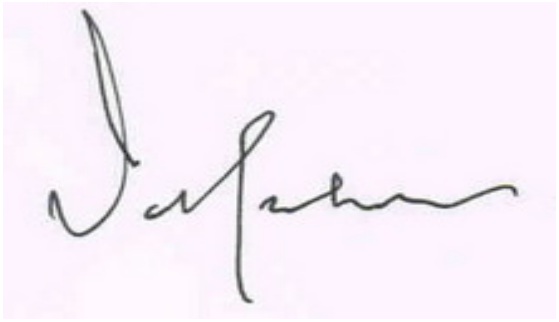
Some Clarifications on Students' Comments

- I notice that some students complained that I was often late for about 15 to 30 minutes in classes. To be exact, they should point out that this incidence happened on Wednesday during our two hours lab sessions starting from 03:30 pm to 05:30 pm. I gathered these students must not have paid attention to my clarification from the beginning of the semester, that I have my family commitment to fulfill as a father, to fetch my three kids home (Taipa) from Sheung Kung Hui Primary School (just around our International Library) every weekday from 03:10 pm. It usually takes me about 20~30 minutes to return to the lab to continue our practical sessions. Most of my students are aware of this personal obligation of mine. Indeed, through blended learning approach (online + face-to-face), in every lab practical, students who came to the lab on time are aware that they need to first finish test-running our tutorial programs (week-by-week) so as to save time for questions and answer when I return. Moreover, I often did extend our lab sessions by the same duration (20~30 minutes) to end well past 05:30 pm.
- I also notice that some students criticize that I never grade their assignments. To be precise, this is not true; yet, I did emphasize throughout the semester that quantitative summative score per exercise or assignment should only come towards the end of the semester, but qualitative comments are rendered formatively week-by-week during the lab session to remind our students the proper way to use our assessment framework, to complete their work, and to perform peer assessment for their fellow students. It is observed that our students are very accounting in terms of score to put in their efforts, rather than concentrating on what to accomplish according to the ILOs announced in the course syllabus. This is a cultural problem for student learning, but we need to be

aware of this, and to encourage our students to get more involved in genuine learning activities instead.

My sincere thanks for your reading my response to the course evaluation of SFTW241, and your attention to the issues brought forth for our discussion during our meeting on 2010DEC07. I am committed to empower our students to get the most of their studies in SFTW241. Their comments and feedback are always welcome.

Sincerely,



Kam Hou VAT
Senior Instructor
Department of CIS – FST
e-mail: fstkhv@umac.mo