SFTW241 Programming Languages Architecture (I)

Kam H. Vat

3.0 credits / 5 weekly contact hours (lectures: 2 hours, lab: 2 hours, tutorial: 1 hour)

COURSE DESCRIPTION

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 computation models covered in this course include the imperative, object-oriented, and concurrent 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. Languages of interest include both software and hardware description languages, such as C versus Verilog, C++ versus Java, Ada versus VHDL. Other contemporary languages considered of interest in inquiry-based learning include: JavaScript versus Ajax, Ruby on Rails versus Groovy on Grails.

SYLLABUS

- PL01: Overview of Programming Languages
- PL02: Virtual Machines
- PL03: Basic Language Translation
- PL04: Declarations and Types
- PL05: Abstraction Mechanisms
- PL06: Object-Oriented Programming

Topics and Intended Learning Outcomes (ILOs)

PL01 – Overview of Programming Languages

Topics:
- History of programming languages
- Brief survey of programming paradigms
- Procedural languages
- Object-oriented languages
- Functional languages
- Declarative, non-algorithmic languages
- Scripting languages
- The effects of scale on programming methodology

Intended Learning Outcomes:
- Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today.
- Identify at least one distinguishing characteristic for each of the programming paradigms covered in this unit
- Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression.
- Distinguish between programming-in-the-large and programming-in-the-small.

PL02 – Virtual Machines

Topics:
- The concept of a virtual machine
- Hierarchy of virtual machines
- Intermediate languages
- Security issues arising from running code on an alien machine

Intended Learning Outcomes:
- Describe the importance and power of abstraction in the context of virtual machines.
- Explain the benefits of intermediate languages in the compilation process.

- Evaluate the tradeoffs in performance versus portability.
- Explain how executable programs can breach computer system security by accessing disk files and memory.

PL03 – Basic Language Translation

Topics:

- Comparison of interpreters and compilers
- Language translation phases (lexical analysis, parsing, code generation, optimization)
- Machine-dependent and machine-independent aspects of translation

Intended Learning Outcomes:

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

PL04 – Declaration and Types

Topics:

- The conception of types as a set of values together with a set of operations
- Declaration models (bindings, visibility, scope, and lifetime)
- Overview of type-checking
- Garbage collection

Intended Learning Outcomes:

- Explain the value of declaration models, especially with respect to programming-in-the-large.
- Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size.
- Discuss type incompatibility.
- Demonstrate different forms of binding, visibility, scoping, and lifetime management.
- Defend the importance of types and type-checking in providing abstraction and safety.
- Evaluate tradeoffs in lifetime management (reference counting versus garbage collection).

PL05 – Abstraction Mechanisms

Topics:

- Procedures, functions, and iterators as abstraction mechanisms
- Parameterization mechanisms (reference versus value)
- Activation records and storage management
- Type parameters and parameterized types
- Modules in programming languages

Intended Learning Outcomes:

- Explain how abstraction mechanisms support the creation of reusable software components
- Demonstrate the difference between call-by-value and call-by-reference parameter passing.
- Defend the importance of abstractions, especially with respect to programming-in-the-large.
- Describe how the computer system uses activation records to manage program modules and their data.

PL06 – Object-Oriented Programming

Topics:

- 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 representation of objects and method tables

Intended Learning Outcomes:

- Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction,

inheritance, and polymorphism.
- Design, implement, test, and debug simple programs in an object-oriented programming language.
- Describe how the class mechanism supports encapsulation and information hiding.
- Design, implement, and test the implementation of “is-a” relationships among objects using a class hierarchy and inheritance.
- Compare and contrast the notions of overloading and overriding methods in an object-oriented language.
- Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
- Describe how iterators access the elements of a container.

FORMAT

Lectures, lab sessions, class and online discussion, small-group mentoring, student oral presentation of project work

ASSESSMENT

Individual Assignments	5%
Pair-Based Assignments	10%
Team-Based Project Assignments	25%
Quizzes	10%
Mid-Term Examination	20%
Final Examination	30%

PREREQUISITES

SFTW120 Programming Science

PEDAGOGY

Dialogic teaching and problem-based learning (PBL): Students are required to complete programming tasks as an individual, as a pair member and as a member of a group. The industrial practice of pair programming is also illuminated throughout the laboratory sessions of the course.

CURRICULUM DESIGN PHILOSOPHY

The SMART Goals Process to improve student learning: Strategic/Specific, Measurable, Attainable, Results-oriented, and Time-sensitive. The materials presented are targeted on students learning outcomes, which are specific, measurable, attainable, and result-oriented within a specific time frame.

SOFTWARE

Open source and vendor-sponsored software on Windows XP: Eclipse 3.3.x, 3.4.x or 3.5.x with CDT, JDT, GNATbench, SimplifIDE VHDL/Verilog, Xpairtise server and client

E-LEARNING PLATFORMS

Moodle 1.9 installed as official internal learning environment at the university. Sakai 2.6.x installed as optional external learning environment for student convenience.

E-RESOURCES

Electronic resources of course materials in previous semesters for each lesson and laboratory are made available to students taking the current course through the UM Moodle environment since 2008. Student coursework materials in terms of project archives are also available through students’ personal e-Portfolios from the UM Moodle’s Blog environment. Other resources such as selected lecture videos currently available since 2007, are made available in the form of DVDs and kept in the University Library's *Reserved Materials* of SFTW241 section for in-library watching on an available-upon-request policy.

REFERENCES

Budd, T.A. (2002). *An Introduction to Object-Oriented Programming*, 3rd edition. New York: Addison Wesley.

Felleisen, M., Findler, R.B., Flatt, M., & Krishnamurthi, S. (2001*). How to Design Program: An Introduction to Programming and Computing*. Cambridge, Mass.: The MIT Press.

Lee, K. (2008). *Programming Languages: An Active Learning Approach*. Berlin: Springer.

Scott, M.L. (2000). *Programming Language Pragmatics*. San Francisco, CA: Morgan Kaufmann.

Sebasta, R.W. (2008). *Concepts of Programming Languages*, 8th edition. New York: Addison Wesley.

Sebasta, R.W. (2008). *Programming the World Wide Web*, 4th edition. New York: Addison Wesley.

Tucker, A.B., & Noonan, R.E. (2007). *Programming Languages: Principles and Paradigms*, 2nd edition. New York: McGraw Hill.

Van Roy, P. & Haridi, S. (2004). *Concepts, Techniques, and Models of Computer Programming*. Cambridge, Mass.: The MIT Press.

Watt, D.A. (2004). *Programming Language Design Concepts*. Chichester, England: John Wiley & Sons.

Watt, D.A. (1991). *Programming Language Syntax and Semantics*. New York: Prentice Hall.

Williams, L. & Kessler, R. (2003). *Pair Programming Illuminated*. Boston, Mass.: Pearson Education, Inc.