MTP

Discuss how a problem may be solved by multiple algorithms, each with

different properties.

• Identify the data components and behaviors of multiple abstract data types.

• Implement a coherent abstract data type, with loose coupling between

components and behaviors.

• Identify the relative strengths and weaknesses among multiple designs or

implementations for a problem.

• Trace the execution of a variety of code segments and write summaries of their

computations.

• Explain why the creation of correct program components is important in the

production of high-quality software.

• Identify common coding errors that lead to insecure programs (eg, buffer

overflows, memory leaks, malicious code) and apply strategies for avoiding such

errors.

• Conduct a personal code review (focused on common coding errors) on a

program component using a provided checklist.

• Describe how a contract can be used to specify the behavior of a program

component.

• Refactor a program by identifying opportunities to apply procedural abstraction.

• Analyze the extent to which another programmer’s code meets documentation

and programming style standards.

• Apply consistent documentation and program style standards that contribute to

the readability and maintainability of software.

Internalize an accurate understanding of what functional and object-oriented programs mean

• Develop the skills necessary to learn new programming languages quickly

• Master specific language concepts such that they can recognize them in strange guises

• Learn to evaluate the power and elegance of programming languages and their constructs

• Attain reasonable proficiency in the ML, Racket, and Ruby languages and, as a by-product, become more

proficient in languages they already know

Course topics:

• Syntax vs. semantics

• Basic ML programming: Pairs, lists, datatypes and pattern-matching, recursion

• Higher-order functions: Lexical scope, function closures, programming idioms

• Benefits of side-effect free programming

• Type inference

• Modules and abstract types

• Parametric polymorphism

• Subtyping

• Dynamically typed functional programming

• Static vs. dynamic typing

• Lazy evaluation: thunks, streams, memoization

• Implementing an interpreter

• Implementing function closures

• Dynamically typed object-oriented programming

• Inheritance and overriding

• Multiple inheritance vs. interfaces vs. mixins

• Object-oriented decomposition vs. procedural/functional decomposition

**KA Knowledge Unit Topics Covered Hours**

PL Object-Oriented Programming

All, with some topics re-enforced from CS1/CS2 (hour count is for just this course)  
PL Functional Programming

PL Basic Type Systems

PL Program Representation

PL Language Translation and Execution

Only these topics are covered: interpretation vs. compilation, run-time

representation of objects and first-class functions, implementation of

recursion and tail calls. The other topics are covered in another required

course.

PL Advanced Programming Constructs

Only these topics are covered: Lazy evaluation and infinite streams,

multiple inheritance, mixins, multimethods, macros, module systems,

“eval”. Exception handling and invariants, pre/post-conditions are

covered in another required course.

PL Type Systems Only these topics are covered (and at only a very cursory level): Type

inference, Static overloading

PL Language Pragmatics

Only this topic is covered: Eager vs. delayed evaluation

Software Development Fundamentals (SDF) 18

Programming Languages (PL) 12

Software Engineering (SE) 12