CSci 658-01: Software Language Engineering
Spring 2018 Lecture Notes

H. Conrad Cunningham

12 February 2018 (after class)

Contents

Professor of Computer and Information Science
University of Mississippi
211 Weir Hall
P.O. Box 1848
University, MS 38677
(662) 915-5358

I maintain these notes as text in the Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the notes to HTML, PDF, and other formats as needed.

Advisory: The HTML version of this document requires use of a browser that supports the display of MathML. A good choice as of February 2018 is a recent version of Firefox from Mozilla.

Lecture Notes

Go To Current Lecture

Schedule, Notes, and Examples

1. **(22 Jan) Review syllabus.** Distribute papers to be discussed over next two weeks.

2. **(24, 26 Jan) Discuss "Little Languages" paper** and related issues

   Jon Bentley. Programming Pearls: Little Languages, *Communications of the ACM*, Vol. 29, No. 8, pp. 711-721, August 1986.

   Instructor's note on `pic`: On my iMac, I installed (using Homebrew) the `groff` package, which includes the GNU `pic` implementation. (I also installed the `plotutils` package to help with display in various formats.)

   Additional references for little language `pic` (not discussed)

   a. Brian W. Kernighan. PIC -- A Graphics Language for Typesetting, Revised User Manual, Computing Science Technical Report No. 116, Bell Laboratories, December 1984. [local]
   b. Eric S. Raymond. Making Pictures with GNU PIC, August 1995.
   c. Philipp K. Janert. In Praise of Pic, *ONLamp.com*, O'Reilly Media, June 2007, Retrieved 24 January, 2018.
   d. W. Richard Stevens. Examples of `pic` Macros, Retrieved from http://www.kohala.com/start/ (troff resources, pic), 24 January 2018. [`pic` source]

   Additional references for little graph language `dot` and the GraphViz package (not discussed)

   e. GraphViz -- Graph Visualization Software, Retrieved 26 January 2018.

   Additional references for the little languages `lex`, `yacc`, `make`, and `chem` (not discussed)

   f. Jon L. Bentley, Lynn W. Jelinski, and Brian W. Kernighan. Chem – A Program for Phototypesetting Chemical Structure Diagrams, AT&T Bell Laboratories, Murray Hill, NJ 07974, U.S.A.
   g. Stuart I. Feldman. Make – A Program for Maintaining Computer Program, *Software: Practice and Experience*, Vol. 9, no. 4 pp. 255-265, 1979.
   h. Stephen C. Johnson. Yacc: Yet Another Compiler-Compiler, Vol. 32. Murray Hill, NJ: Bell Laboratories, 1975.
   i. Michael E. Lesk and Eric Schmidt. Lex: A Lexical Analyzer Generator, Bell Laboratories Murray Hill, NJ, 1975.
   j. Learn X in Y Minutes, Where X = make (GNU)

3. **(26, 29 Jan) Discuss "No Silver Bullet" paper** and related issues

   Frederick P. Brooks. No Silver Bullet: Essence and Accident in Software Engineering, *IEEE Computer*, Vol. 20, No. 4, 10-19, 1987.

4. **(29, 31 Jan) Discuss "Language-Oriented Programming" paper**

   M. P. Ward. Language-Oriented Programming, *Software–Concepts and Tools,* Vol. 15, No. 4, pp. 147-161, 1994. [local]

   Other references from paper (not discussed)

   a. Charles Antony Richard Hoare. The Emperor's Old Clothes (1980 Turing Award Address), *Communications of the ACM*, Vol. 24, No. 2, 75-83, 1981.

5. **(31 Jan; 2, 5, 14 Feb) Discuss Domain Specific Languages notes** and related issues

   References used in notes (not discussed directly)

   a. Martin Fowler. Using Domain-Specific Languages, Chapter 2, *Domain-Specific Languages*, Addison Wesley, 2011.
   b. Steve Freeman and Nat Pryce. Evolving an Embedded Domain-Specific Language in Java, In *Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 12 pages, 2006. [local]
   c. Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler and Steven Voelkel. Design Guidelines for Domain Specific Languages, In *Proceedings of OOPSLA Workshop on Domain-Specific Modeling*, 2009. Also arXiv preprint arXiv:1409.2378, 2014. [local]
   d. M. Mernik, J. Heering, and A. M. Sloane. When and How to Develop Domain Specific Languages, *ACM Computing Surveys*, 37(4):316-344, December 2005. [local]

6. **(5, 14 Feb) Discuss Fowler's introductory DSL example**

   Martin Fowler. An Introductory Example, Chapter 1, *Domain-Specific Languages*, Addison Wesley, 2011.

7. **Survey Design Patterns** (see dates on items)

   a. **(5, 7 Feb)** Introduction to Patterns [HTML slides]

   b. **(7 Feb)** Pipes and Filters Architectural Pattern (Not fully covered, used mainly to illustrate approach to pattern definition) [Powerpoint]

   c. **(9, 12 Feb)** John Vlissides. Designing with Patterns, In *Pattern Hatching: Design Patterns Applied*, Addison-Wesley, 1998. [slides]

   d. Additional references from instructor's notes (not directly disucssed in class)

      - Factory Method Design Pattern (Powerpoint)
      - Strategy Design Pattern (Powerpoint)
      - Template Method Design Pattern (Powerpoint)

e. Additional references (mentioned in class, but not discussed in depth)

- [Siemens book] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley, 1996.
- ["Gang of Four" (GoF) book] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995. Patterns*, Wiley, 1996.
- Mary Shaw. Some Patterns for Software Architecture, In John M. Vlissides, James O. Coplien, and Norman L. Kerth, editors, *Pattern Languages of Program Design 2*, Addison Wesley, 1996, pages 255-270. [local]

8. Reference material possibly useful at this point (may discuss explicitly later if needed)

   a. Python 3

   - Learn X in Y Minutes, Where X = Python3

   b. Scala

   - Learn X in Y Minutes, Where X = Scala
   - Notes on Scala for Java Programmers
   - Paul Chiusano and Runar Bjarnason. *Functional Programming in Scala*, Manning, 2015.
     - Notes on Functional Data Structures (Chapter 3)
     - Notes on Error Handling without Exceptions (Chapter 4)
     - Notes on Strictness and Laziness (Chapter 5)
   - Martin Odersky. *Scala by Example*, EPFL, 2014. [local]
   - Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala*, First Edition, Artima, 2008.

   c. Object-oriented software development

   - Object-Oriented Software Development
   - Programming Paradigms (especially sections on the object-oriented and prototype-based paradigms)
   - Using CRC Cards (Class-Responsibility-Collaboration)
   - Kent Beck and Ward Cunningham. A Laboratory for Teaching Object-Oriented Thinking, In *Proceedings of the OOPSLA'89 Conference*, ACM, 1989.
   - Paul Gestwicki. CRC Card Analysis (YouTube), Ball State University, February 2016.

   d. Free online programming language textbooks and tutorials

9. **(2, 5, 14 Feb) Discuss State Machine DSLs** based on Martin Fowler's Secret Panel Controller (State Machine) DSLs

Caveat: I developed the Scala versions in 2009 by referring to an in-work version of Fowler's book on his website, so they may not follow his final book precisely. Fowler's code is in Scala, Ruby, C#, etc. I updated the Scala programs in 2018 to make sure they would compile under the current Scala.

a. Background: Martin Fowler. *Domain Specific Languages*, Addison Wesley, 2011.

   - Fowler introduces the Secret Panel Controller (State Machine) case study in Chapter 1 (An introductory example) It is an example running through several of the 57 chapters of the book.
   - Martin Fowler's List of DSL Patterns from *Domain Specific Languages*, Addison Wesley, 2011

   Scala versions (2009)

b. Scala State Machine Semantic Model (Ch. 1, 3, 11)

   - StateMachine.scala
   - StateMachineTest.scala
   - CommandChannel.scala (mock)
   - StateMachineDirect.sh test script

c. Scala XML-based External DSL (Ch. 1, 3, 5)

   - StateMachineXMLTest .scala
   - IncrementalStateMachineBuilder.scala
   - input file SecretPanel.xml
   - StateMachineXML.sh test script

d. Scala Custom External DSL with a Delimiter-Directed parser (Ch. 1, 3, 5, 17)

   - DelimiterDSLTest.scala
   - IncrementalStateMachineBuilder.scala
   - input file CustomExternalStateMachineDSL.dsl
   - DelimiterDSL.sh test script

e. Scala Custom External DSL with hand-coded Ad Hoc Recursive-Descent parser (Ch. 1, 3, 5, 21)

   - RecursiveDescent.scala
   - IncrementalStateMachineBuilder.scala
   - input file CustomExternalStateMachineDSL2.dsl
   - RecursiveDescent.sh test script

f. Scala Custom External DSL (using Scala parser combinator library) with embedded state machine builder (Ch. 1, 3, 5, 22)

   - CombinatorParserBuilderTest.scala

- IncrementalStateMachineBuilder.scala
- input file CustomExternalStateMachineDSL2.dsl
- CombinatorParserBuilder.sh test script

g. Scala Custom External DSL (using Scala parser combinator library) with full AST construction (Ch. 1, 3, 5, 22)

- CombinatorParserASTTest.scala
- input file CustomExternalStateMachineDSL2.dsl
- CombinatorParserAST.sh test script

h. Scala Static C Code Generator with Model-Aware target platform library (Ch. 1, 3, 5, 8, 52, 55)

- StaticC_GeneratorTest.scala
- generated C output file output.c
- StaticC_Generator.sh test script
- Note: The needed framework code needed to run the generated C program is not yet available in this form. It needs to be reconstructed from Fowler's book.

10. **(5, 14 Feb) Discuss Fowler's Computer Configuration Internal DSLs** (Ch. 4, 35, 36, 38)

    a. Background: Martin Fowler. *Domain Specific Languages*, (Addison Wesley, 2011).

    Scala versions (2009)

    b. Scala semantic model (shared)

    c. Scala internal DSL using Method Chaining

    d. Scala internal DSL using Nested Closures and Object Scoping

    e. CompConfig.sh test script

11. (TBD) Discuss Fowler's Email Message Building Internal DSL (Ch. 35)

    a. Background: Martin Fowler. *Domain Specific Languages*, (Addison Wesley, 2011).

    Scala versions (2009)

    b. Scala internal DSL using Method Chaining and Progressive Interfaces

    c. EmailProgressive.sh test script

12. **(assigned, not discussed yet, TBD) Discuss Coplien's SCV paper**

    James Coplien, Daniel Hoffman, and David Weiss. Commonality and Variability in Software Engineering, *IEEE Software*, Vol. 15, No. 6, November 1998.

Materials under Phase 2 Reconstruction

101. (TBD) Domain Modeling slides, for course registration system example.

102. (TBD) Mark Ardis, Nigel Daley, Daniel Hoffman, Harvey Siy, and David Weiss. Software Product Lines: A Case Study, *Software Practice and Experience*, Vol. 30, No. 7, pp. 825-847, 2000.

103. (TBD) S. Thibault, R. Marlet, and C. Consel. Domain-Specific Languages: From Design to Implementation–Application to Video Device Drivers Generation. *IEEE Transactions on Software Engineering*, 25(3):363–377, May/June 1999.

104. (TBD) Ralph E. Johnson and Brian Foote. Designing Reusable Classes, *Journal of Object-Oriented Programming*, Vol. 1, No. 2, pages 22-35, June/July 1988.

105. (TBD) Develop a Survey DSL

    a. Background: H. Conrad Cunningham. A Little Language for Surveys: Constructing an Internal DSL in Ruby, In *Proceedings of the ACM SouthEast Conference*, 6 pages, March 2008.

        - manuscript
        - presentation

    b. Ruby source code (2006, 2008)

        - Ruby source code SurveyLanguage.rb
        - test DSL input file
        - test DSL input file with errors

106. (reference material on Lua, from Fall 2016 CSci 450 notes)

    a. Reference: Roberto Ierusalimshcy. *Programming in Lua*, Fourth Edition, Lua.org, Rio de Janiero, Brazil, 2016. The First Edition of this book is available online at https://www.lua.org/pil/contents.html; it covers Lua 5.0.

    b. Background: The slides below are adapted, in part, from *Programming in Lua, Slides for Course* by Fabio Mascarenhas from the Federal University of Rio de Janeiro, Brazil. He taught a course, which used Lua 5.2, at Nankai University, P. R. China, in July 2013.

    c. Introduction to Lua slides based, in part, on Mascarenhas slide sets 0-5

    d. Advanced Lua Functions slides based, in part, on Mascarenhas slide set 6

    e. Modules in Lua slides based, in part, on Mascarenhas slide set 9

    f. Lua Metatables slides based, in part, on Mascarenhas slide set 10

g. Lua Objects slides based, in part, on Mascarenhas slide set 11

107. (TBD) Discuss Fowler's Lair Configuration DSL

Caveat: Some of these DSL programs may depend upon features from Lua 5.1 that were changed in Lua 5.2 and 5.3.

a. Background:

- Case study: Martin Fowler's One Lair and Twenty Ruby DSLs, Chapter 3, *The ThoughtWorks Anthology: Essays on Software Technology and Innovation*, The Pragmatic Bookshelf, 2008 [local chapter]
- Patterns (repeated from above): Martin Fowler's List of DSL Patterns from *Domain Specific Languages*, Addison Wesley, 2011
- Source code: [entire book] [Fowler's Ruby DSLs]

Lua versions (2013)

b. Lua shared modules

- Class support module (for implementing classes in Lua)
- Semantic model
- Test driver for semantic model (rules00.lus)

c. Lua internal DSLs

- Using Global Function Sequence pattern
  builder module (builder08.lua) – dsl script (rules08.lua) – test driver (test08.lua)
- Using Class Method Function Sequence and Method Chaining patterns
  builder module (builder11.lua) – dsl script (rules11.lua)
- Using Expression Builder and Method Chaining patterns
  builder module (builder14.lua) – dsl script (rules14.lua) – test driver (test14.lua)
- Using Nested Closures pattern
  builder module (builder03.lua) – dsl script (rules03.lua) – test driver (test03.lua)
- Using Expression Builder, Object Scoping, and Method Chaining patterns
  builder module (builder17.lua) – dsl script (rules17.lua) – test driver (test17.lua)
- Using Literal Collection pattern
  builder module (builder22.lua) – dsl script (rules22.lua) – test driver (test22.lua)

d. Lua external DSLs

- Using LPEG Parser/Builder (no corresponding example in Fowler paper)
  builder module (builderLPEG1.lua) – dsl script (rulesLPEG1.dsl) – test driver (testLPEG1.lua)

108. (TBD) Discuss Fowler's DSL Reader framework

    a. Background:

    - Martin Fowler. Language Workbenches: The Killer-App for Domain Specific Languages? June 2005.
    - Martin Fowler. Generating Code for DSLs,, June 2005.

    Ruby DSLs (2006)

    b. Ruby shared modules

    - DSL Reader Framework module
    - DSL Reader Utilities mix-in module
    - Data input file
    - Text DSL description
    - XML DSL description

    c. Ruby direct configuration and testing of Reader

    - BuilderDirect.rb

    d. Ruby single-pass external text DSL

    - TextSinglePass.rb

    e. Ruby two-pass external XML DSL

    - TwoPass.rb
    - class BuilderExternal source code generated by TwoPass.rb

    f. Ruby internal DSL

    - RubyDSL.rb

109. (TBD) Sandwich DSL Case Study

    a. Haskell version (2017)

    b. Scala version (2016)

    c. Lua version (simpler, 2013)

    - Semantic model module
    - DSL builder module using function sequence pattern
    - Test driver

110. (TBD) Abstract data types and modular design

    a. Background: Abstract data types

- Nell Dale and Henry Walker. Abstract specification techniques, Chapter 1, In *Abstract Data Types: Specifications, Implementations, and Applications*, pp. 1-34,D. C. Heath, 1996.
- H. Conrad Cunningham, Yi Liu, and Jingyi Wang. Designing a flexible framework for a table abstraction, Chapter 13 in Y. Chan, J. Talburt, and T. Talley, editors, *Data Engineering: Mining, Information, and Intelligence*, pp. 279-314, Springer, 2010. [manuscript] [slides]

b. Modular Design

c. Data Abstraction: [slides]

d. (OO reference materia, repeated from abovel) Object-Oriented Software Development

e. (Programming paradigms reference material, repeated from above) Programming Paradigms

f. William R. Cook. On understanding data abstraction revisited. In *Proceedings of OOPSLA*, October 2009.
[local]

g. K. H. Britton, R. A. Parker, and D. L. Parnas. A procedure for designing abstract interfaces for device interface modules, In *Proceedings of the 5th International Conference on Software Engineering*, pp. 195-204, March 1981.

h. D. L. Parnas. On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, Vol. 15, No. 12, pp. 1053-1058, 1972.

i. D. L. Parnas. On the design and development of program families, *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, pp. 1-9, March 1976.

j. D. L. Parnas. Designing software for ease of extension and contraction, *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 1, pp. 128-138, March 1979.

k. D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems, *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 3, pp. 259-266, March 1985.

Materials under Phase 1 Reconstruction

201. (TBD) Examine the CookieJar ADT case study (in Scala)

    a. Cookie Jar ADT Problem Description

b. Specification concepts and notation – from Candy Bowl ADT description

c. Immutable CookieJar ADT Implementation in Scala (`ICookieJar`) – uses method chaining functional style with immutable objects

- ADT specification as Scala trait
- List version
- HashMap version
- List of tuples version
- Blackbox test script

d. Mutable CookieJar ADT Implementation in Scala (`CookieJar`) – uses object-oriented style with mutable state

- ADT specification as Scala trait
- [List version ArrayBuffer version
- HashMap version
- List of tuples version
- Array version
- Blackbox test script

e. Similar specification and Ruby program

202. (TBD) Examine Carrie's Candy Bowl ADT (abstract data type) case study

a. Lua

- ADT Semantics
- Hashed version
- Unsorted List version
- Test driver

b. Scala

- CandyBowl trait
- CandyBowlList class

203. (TBD) Examine the Labeled Digraph case study

a. Background:

- Nell Dale and Henry Walker. "Directed Graphs or Digraphs," Chapter 10, In *Abstract Data Types: Specifications, Implementations, and Applications*, pp. 439-469, D. C. Heath, 1996.
- Conrad Barski. "Building a Text Game Engine,", Chapter 5, In *Land of Lisp: Learn to Program in Lisp, One Game at a Time*, pp. 69-84, No Starch Press, 2011.

  The Common Lisp example in this chapter is similar to the classic Adventure game; the underlying data structure is a labeled digraph.

b. Haskell solutions (2015)

- Labelled Digraph Abstract Data Type
- List representation for vertices and edges: [module] [test module]
- Map representation for graph: [module] [test module]

c. Elixir solutions (2015)

- Tuple and list representation for graph: [module] [test module]

d. Scala solutions (2016)

- Abstract data type specification and Scala trait (interface): [ADT interface]
- List representation for vertices and edges: [class source] [test source]
- Map (HashMap) representation for graph: [class source] [test source]

e. Using the Elixir Digraph ADT module to build the Wizard's Adventure game (2015)

- Wizard's Adventure game, Version 1, adapted from Chapter 5, 6, and 17 of Conrad Barski's *Land of Lisp: Learn to Program in Lisp, One Game at a Time*, No Starch Press, 2011.
- Wizard's Adventure game, Version 2, that uses a higher order function to generate game actions and improved handling of the game state.

204. (TBD) Parsing Expression Grammars

a. Wikipedia entry on Parsing Expression Grammar

b. Bryan Ford. Parsing Expression Grammars: A recognition-based syntactic foundation.
[slides]

205. (TBD) Doug Rosenberg with Kendall Scott. "Domain Modeling," Chapter 2 in *Use Case Driven Object Modeling with UML*, Addison Wesley 1999.

206. (TBD) Doug Rosenberg with Kendall Scott. "Use Case Modeling," Chapter 3 in *Use Case Driven Object Modeling with UML*, Addison Wesley 1999.

207. (TBD) Reference: Alistair Cockburn. "Introduction," Chapter 1 in *Writing Effective Use Cases*, Addison-Wesley, 2001.

208. (TBD) Understanding Inheritance, based on Timothy Budd's *Understanding Object-Oriented Programming with Java*, Chapter 8:
[slides]

209. (TBD) Simple, silly Employee hierarchy example

a. Scala (2008, 2010)

b. Ruby (2006)

210. (TBD) Examine a natural number arithmetic package

This case study has implementations in four different languages (with some slight differences among the examples). So it can be used to compare implementations in different languages.

a. Background on Peano arithmetic

- Peano Axioms, Wikipedia article
- Peano's Axioms, Wolfram MathWorld article

b. Background on software design patterns

- Source Making website

c. Lua version (2013)

d. Elixir version (2015)

- Nat module
- test module
- test script

e. Scala versions (2012, 2016)

- Functional object-oriented style with ordinary classes
- Functional object-oriented style with case classes
- Functional module style with case classes

f. Ruby version (2006)

g. Java version (2004, 2016), simpler, no generics

- abstract base class Nat
- subclass Zero
- subclass Succ
- subclass Err
- TestNat main program

211. (TBD) Software Reuse, based on Timothy Budd's *Understanding Object-Oriented Programming with Java*, Chapter 10:
]slides]

- Scala translation of Frog dynamic composition example

212. (TBD) Replacement and Refinement, loosely based on Timothy Budd's *An Introduction to Object-Oriented Programming*, Third Edition, Section 16.2:
[slides]

213. (TBD) Implications of Inheritance, based on Timothy Budd's *Understanding Object-Oriented Programming with Java*, Chapter 11:

[slides]

214. (TBD) Multiple Inheritance, based on Timothy Budd's *An Introduction to Object-Oriented Programming*, Third Edition, Chapter 13:
[slides]

    - [Scala Modified Philosophical Frog example from Odersky et a [source]
    - Scala Modified Stackable traits example (IntQueue) from Odersky et al: [source]

215. (TBD) Polymorphism, based on Timothy Budd's *Understanding Object-Oriented Programming with Java*, Chapter 12:
[slides]

216. (TBD) Second Look at Classes, loosely based on Timothy Budd's *An Introduction to Object-Oriented Programming*, Chapter 25 on Reflection and Introspection:
[slides]

217. (TBD) Frameworks, based on Timothy Budd's *An Introduction to Object-Oriented Programming*, Third Edition, Chapter 21.

    a. Simple Sorting Framework in Scala

        - Low-level concrete Employee sorting
        - Insertion Sorting framework
        - Employee Sorting application of framework
        - Test code for Employee Sorting application of framework

    b. Ice Cream Store discrete event simulation

        - Simulation framework
        - Ice Cream Store application

218. (TBD) Scala Divide-and-Conquer Framework, similar to the Java framework in the paper:

    H. C. Cunningham, Y. Liu, and C. Zhang. Using classic problems to teach Java framework design, *Science of Computer Programming*, Special Issue on Principles and Practice of Programming in Java (PPPJ 2004), Vol. 59, No. 1-2, pp. 147-169, January 2006. doi: 10.10.16/j.scico.2005.07.009. [manuscript]

    Note: The above paper was not discussed in class, but the Scala versions of the Divide-and-Conquer (immediately below) and Binary Tree Traversal (farther down on this page) frameworks were discussed.

    - Template-based Divide-and-Conquer Framework (DivConqTemplate)
    - Strategy-based Divide-and-Conquer Framework (DivConqStrategy)

- Traits for Problem and Solution descriptions for both frameworks (DivConqProblemSolution)
- Application of Template-based framework to QuickSort (QuickSortTemplateApp)
- Application of Strategy-based framework to QuickSort (QuickSortStrategyApp)
- Descriptor for QuickSort state for both QuickSort applications (QuickSortDesc)

219. (TBD) Introduction to Design Patterns, Chapter 6 in, Eric Braude, *Software Design: From Programming to Architecture*, Wiley, 2004.

220. (TBD) [Siemens book] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley, 1996.

221. (TBD) ("Gang of Four" (GoF) book) Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

222. (TBD) Mark Grand. *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML*, Volume 1, Wiley, 1998.

    - Template Method Design Pattern: [slides]
      Reference: Gof, "Template Method," pp. 325-329.
      Reference: Grand, "Template Method," pp. 377-384.
    - Strategy Design Pattern: [slides]
      Reference: GoF, "Strategy," pp. 315-323.
      Reference: Grand, "Strategy," pp. 371-376.
    - Factory Method Design Pattern: [slides]
      Reference: GoF, "Factory Method," pp. 107-116.

223. (TBD) Binary Tree Visitor Scala Framework

    a. Background: H. C. Cunningham, Y. Liu, and C. Zhang. Using classic problems to teach Java framework design, *Science of Computer Programming*, Special Issue on Principles and Practice of Programming in Java (PPPJ 2004), Vol. 59, No. 1-2, pp. 147-169, January 2006. doi: 10.10.16/j.scico.2005.07.009.
    [manuscript]

    b. Additional pattern references

        - GoF, "Composite," pages 163-174.
          Grand, "Composite," pages 165-174.
        - GoF, "Visitor," pages 331-344.
          Grand, "Visitor," pages 385-395.

    Scala versions (2008, 2010)

c. Straightforward translation of the non-generic Java program to Scala

- Top-level framework (BinTreeFramework)
- Second-level Euler tour framework (EulerTourVisitor)
- Second-level mapping visitor framework (MappingVisitor)
- Second-level breadth-first framework (BreadthFirstVisitor)
- Application of BinTree frameworks (BinTreeTest)

d. Generic implementation of BinTree framework in Scala.

- Top-level framework (BinTreeFramework)
- Second-level Euler tour framework (EulerTourVisitor)
- Second-level breadth-first framework (BreadthFirstVisitor)
- Application of BinTree frameworks (BinTreeTest)

224. (TBD) Use the Digraph ADT module to build the Wizard's Adventure game

a. Wizard's Adventure game, Version 1, adapted from Chapter 5, 6, and 17 of Conrad Barski's *Land of Lisp: Learn to Program in Lisp, One Game at a Time*, No Starch Press, 2011.

b. Wizard's Adventure game, Version 2, that uses a higher order function to generate game actions and improved handling of the game state.

225. (TBD) Examine the Dice of Doom game.

a. Background: Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time*, No Starch Press, 2011.

b. Dice of Doom, Version 1a, is a basic eagerly evaluated version (similar to that developed on pages 303-325 of *Land of Lisp*). This version supports either two human players or a human player and a simple, minimax search-based "AI" (artificial intelligence) opponent.

c. Dice of Doom, Version 1b, is an eager version above with an attempt at memoization of functions `neighbors` and `game_tree` using the Elixir `Agent` modules.

d. Dice of Doom, Version 2a, is a lazy version using Elixir `Stream` data structures (but without the memoization optimizations). This is based on version 1a above plus the discussion from pages 384-389 in chapter 18 of *Land of Lisp*. This version implements a limited depth minimax search, but it does not implement the artificial intelligence heuristics given in the last part of chapter 18.

226. (TBD) Arithmetic expression tree program skeletons

a. Background: These are expanded from the example in Notes on Scala for Java Programmers.

Lua versions (2013-16)

b. Lua recursive function versions (2013, 2014)

- Lua Recursive Functions with Record Representation
- Lua Recursive Functions with List Representation
- Lua Evaluation Function Table with List Representation

c. Lua object-oriented versions (2013, 2016)

- Lua Prototype Object-Based
- Lua Object-Oriented with Inheritance

d. Lua LPEG parsers (2013)

- Parser with captures
- Parser with semantic actions

Scala versions (2008-16)

e. Scala recursive function version using case classes

f. Scala traditional object-oriented version

227. (TBD) Study the Movable and Named Objects case study (in Lua but based on a Haskell case study by Thompson)

a. *Purpose*: The Named and Movable Objects case study explores use of inheritance hierarchies and multiple inheritance in Lua. I also extracted a "class support module" from the initial version of this code; this module was used in later case studies (e.g., Lair Configuration DSL).

The "makeClass" function dynamically creates a "class" that has zero or more superclasses. It creates the class prototype object with an appropriate constructor/initialization function, appropriate default definitions of instance methods, and the needed settings of metaclasses and the `__index` metamethod to support the inheritance hierarchy.

b. Background reading on object-oriented programming languages: Object-oriented subsection of the Fundamental Concepts of Programming Languages notes

c. Background reading on case study: Section 14.6 of Simon Thompson. *Haskell: The Craft of Functional Programming*, Third Edition, Addison Wesley, 2011

Note: We should readdress the design and implementation of this case study and the class support module. The latter may be too complex for our purposes in this course.

d. First Lua version (movable.lua)

e. Modularized Lua version with improved class support:

- class support module (class_support.lua)

- module using class-support (movable2.lua)
- test driver (movable2Test.lua)

f. Other older versions:

- Haskell source
- Scala source (partial)

228. (TBD) Examine the complex number arithmetic modules in Lua

Modules are repeated in each package in which they are used

a. *Background* reading: *Structure and Interpretation of Computer Programs*, Second Edition, MIT Press, 1996, Section 2.4

SICP video lectures, Hal Abelson

b. Rectangular coordinates modules:
[arithmetic] [rectangular representation] [utilities] [test driver]

c. Polar coordinates modules:
[arithmetic] [polar representation] [utilities] [test driver]

d. Tagged data modules: [arithmetic] [data tagging] [utilities] [test driver]

e. Data-directed programming modules:
[arithmetic] [rectangular representation] [polar representation] [data tagging] [utilities] [test driver]

f. Object-oriented modules:
[arithmetic] [utilities] [test driver]

229. (TBD) Kamin Interpreters in Lua Toolset (KILT)

a. Kamin-Budd Interpreters

b. Language/Interpreter-independent modules:
[REPL Module (repl.lua)]
[Environment Module (environment.lua)]
[Function Table Module (funtab.lua)]
[Utilities Module (utilities.lua)]
[Opcodes Factory Module (opcodes.lua)]
[Values Factory Module (values.lua)]
[Parser Factory Module (parser.lua)]
[Evaluator Factory Module (evaluator.lua)]

c. Kamin Chapter 1 Core language interpreter:
[Core Interpreter (Core.lua)]
[Core Opcodes (opcodes_core.lua)]
[Core Values Module (values_core.lua)]
[Core Parser Module (parser_core.lua)]
[Core Evaluator Module (evaluator_core.lua)

d. Kamin Chapter 2 Lisp language interpreter:
[Lisp Interpreter (Lisp.lua)]
[Lisp Opcodes (opcodes_lisp.lua)]
[Lisp Values Module (values_lisp.lua)]
[Lisp Parser Module (parser_lisp.lua)]
[Lisp Evaluator Module (evaluator_lisp.lua)
[A few Lisp examples]

e. Kamin Chapter 4 Scheme language interpreter:
[Scheme Interpreter (Scheme.lua)]
[Scheme Opcodes (opcodes_scheme.lua)]
[Scheme Values Module (values_scheme.lua)]
[Scheme Parser Module (parser_scheme.lua)]
[Scheme Evaluator Module (evaluator_scheme.lua)
[A few Scheme examples]

230. (TBD) Lua list module case study (Cell List)

Purpose: This Lua case study illustrates (i) functional programming principles, (ii) design and implementation methods for abstract data types and information hiding modules, and, (iii) Lua programming techniques (linked lists, stateless iterators, closures, metatables, etc.)

Background reading on Lua: Chapter 11 on data structures (pages 107-116) and Chapter 15 on modules (pages 151-161) of *Programming in Lua* (PiL), Third Edition

Caveats: (1) These modules (from 2013-14) internally layer the operations into primitive and non-primitive operations, but they do not separate the primitive operations into its own module. That can be done similarly to the Rational Arithmetic case study. (2) In the future, I plan to construct a more efficient implementation that uses array-style tables. The table-based versions likely also need to use weak tables to avoid memory leaks.

a. Cell-based list module:
[module source] [test driver]

b. Closure-and-table-based list module variant:
[module source] [test driver]

c. Function-based cell list module variant:
[module source] [test driver]

d. Lazy list module variant using C preprocessor (cpp -P):
[module source] [source after cpp]
[test driver] [driver after cpp] [sh script]

e. Lazy list module variant using Lua Macro 2.5:
[macro definitions] [module macro source] [source after luam -o]
[macro test driver] [driver after luam -o] [sh script]

231. (TBD) Examine functions adapted from SICP

    a. Background reading: Chapter 1 of the classic textbook SICP – Harold Abelson and Gerald J. Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs*, Second Edition, MIT Press, 1996:
    [book site at MIT Press] [HTML] [SICP ebook site

    b. First-order functions in Scala

        - Square root (Newton's Method) with all public functions
        - Square root (Newton's Method) with nested function definitions
        - Factorial
        - Fibonacci
        - Exponentiation
        - Greatest common divisor

    c. Higher-order functions in Scala

        - Summation (takes function arguments)
        - Derivative (returns function result)

    d. Lua versions

        - Square root (Newton's Method)
        - Factorial
        - Fibonacci
        - Exponentiation
        - Greatest common divisor
        - Summation
        - Derivative

    e. Also various Haskell, Elixir, Elm, and Scheme versions

Unprocessed Old Materials

301. Notes on Recursion Concepts and Terminology

302. Notes on Functional Program Evaluation Concepts

303. Jim Weirich. "Rake Tutorial," "Rake User Guide," and "Rake by Example," 2005.
(Basic ideas discussed.) [Ruby Rake web site]

304. Jim Freeze. Creating DSLs with Ruby, Artima Developer, http://www.artima.com/rubycs/articles/ruby/_as/_dsl.html, March 2006.
[Artima website] [[local printer-friendly copy](http://www.cs.olemiss.edu/~hcc/engr692ruby/notes/ruby/_as/_dslP.html)]

305. Jamis Buck. Writing Domain Specific Languages, `the { buckblogs :here }`, 20 April 2006.
[blog entry]