CSU 670

September 8, 2006

Software Development — Syllabus
Professor Karl. J. Lieberherr

The syllabus is subject to change based on class reaction.

`http://www.ccs.neu.edu/research/demeter/course/topics-covered/topics-covered`

contains a good overview of computer science topics covered in this course. Six of nine important areas are touched. Three fundamental computer science processes are touched: theory, abstraction and design. And 5 of 12 recurring computer science concepts are touched.

This course provides state-of-the-art techniques and concepts for software development with a focus on proper separation of concerns. We will review the history of software development and encounter different techniques for separation of concerns like functions and objects. We will identify limitations in current software development practice that lead to bad separation of concerns. We will touch on general-purpose aspect-oriented techniques that lead to better separation of concerns. Then we will identify limitations in those general-purpose techniques and point to special purpose aspect-oriented techniques. We will use the Demeter Method as an example of a special purpose aspect-oriented technique.

The goal of proper separation of concerns in software development is to make programs look like designs.

The course will also cover the people skills needed in agile software development. How do you write requirements so that customers can understand them; how to do small iterations in the development cycle to give customers feedback; how to do design reviews with your peer software developers; etc.

The course will also teach you strong pattern matching skills that are needed for working with abstractions and proper separation of concerns. You will match design patterns and idioms with Java programs, object graphs with class graphs and traversal specifications with class graphs and object descriptions with class dictionaries, an extended form of class graphs. Those pattern matching skills are useful in many other contexts than software development.

The course has also a mathematical component where we will work with undecidable problems and suitable approximations. We will work with finite non-deterministic automata and their efficient implementation. That is why the course has Theory of Computation as a prerequisite.

Designing and programming will be done in a structure-shy, grammar-based, object-oriented style. All assignments and the project will be written directly or indirectly in Java.

This course does not have Java as a prerequisite, however you are expected to learn a small subset of Java from a good Java book of your choice.

The lectures gradually introduce you to programming adaptively. First we program adaptively in pure Java using the DJ library. Hw 1 and hw 2 exercise this knowledge. In later homeworks and the project we use a small extension to Java to write our adaptive programs as behavior files. Structure will be defined by class dictionaries similar to XML schemas. Behavior files allow us in many cases to keep information about one concern in one file. The code will be spread automatically into multiple Java classes.

1. Week: History of software development techniques. Introduction to agile software development. Separation of concerns example: bus system simulation. Understanding a program without documentation.

    - Pattern: Structure-shy Traversal.
      http://www.ccs.neu.edu/research/demeter/adaptive-patterns/pattern-lang-conv file: Structure-Shy-Traversal.html
    - Pattern: Selective Visitor.
      http://www.ccs.neu.edu/research/demeter/adaptive-patterns/pattern-lang-conv file: Selective-Visitor.html
    - Pattern: Class Graph (Class dictionaries and Unified Modeling Notation.)
      http://www.ccs.neu.edu/research/demeter/adaptive-patterns/pattern-lang-conv file: Class-Graph.html
    - Overview of pattern language, see: http://www.ccs.neu.edu/research/demeter/adaptive-patterns/pattern-lang-conv file: pattern-language-for-AP.html

    AP book chapters: selections from 1-4. TPP: The Law of Demeter section 26: Decoupling and the Law of Demeter.

2. Week: Requirements Engineering

    - Pattern: Structure-Shy Object
      http://www.ccs.neu.edu/research/demeter/adaptive-patterns/pattern-lang-conv file: Structure-Shy-Object.html
    - Pattern: Growth Plan
      http://www.ccs.neu.edu/research/demeter/adaptive-patterns/pattern-lang-conv file: Growth-Plan.html

Writing simple adaptive programs using traversals and visitors. Introduction to DJ. Class dictionary design.

AP book chapters: selections from 5-8. TPP Chapter 2: A Pragmatic Approach (The Evils of Duplication (DRY), Orthogonality (Separation of Concerns, AOP), Reversibility (can you change your design decisions), Tracer Bullets (incremental software development using a growth plan), Prototype (leraning), Domain Languages (an application of class dictionaries), Estimate (iterate the schedule with the code)).

3. Week: Importance of testing and robust test cases. Strategy graphs. Decoupling classes: Law of Demeter. Relationship to adaptive software. DJ continued. The class dictionary notation (graphical and textual). Design rule checking of class dictionaries. Class dictionaries as customizers for adaptive programs.

   AP book chapters: selections from 8-11. TPP Section 27: Metaprogramming (put abstractions in code, details in metadata)

4. Week:

   Visitor pattern. Adaptive Programming. Improving the reusability of software designs. Parameterized class definitions.

   AP book chapters: selections from 10-12. TPP: Section 21: Design by Contract.

5. Week: Traversal strategies in detail. The Demeter Method: Talk only to your friends that share the same concerns. Design notations for behavior. Growth Plan pattern. Developing a growth plan for implementation and testing. Design with maintenance in mind.

   AP book chapters: 13. TPP: Section 29: It is just a view (separate views from models: how is this used in Eclipse?)

6. Week: Midterm. Testing of adaptive, object-oriented software. Eclipse.

   TPP: section 42 Ubiquitous Automation and section 43: Ruthless Testing.

7. Week: Class dictionary design and class dictionary transformations. The 11 kinds of class dictionaries. Project program design and implementation. Eclipse (continued).

8. Week: Project program design and implementation. Eclipse (continued).

9. Week: Project program design and implementation. Eclipse (continued). Further design patterns for aspect-oriented software development.

10. Week: A comparison of object-oriented software development methodologies.

11. Week: Design Reviews.

12. Week: Design Reviews.

13. Week: Design Reviews.

14. Week: Design Reviews.

15. Exam week: Final.