The UNIVERSITY of VERMONT

Benjamin Fenster
CS-205 (Damon)
20 December 2004

Code Design

- I. Introduction
  - a. This is the next step after working out the system design.
  - b. Obviously will have to go back and rethink some of the system design, but the high-level design will be *mostly* done.
  - c. Doesn’t tell you how components work, just what they do
  - d. Three Big Aspects
    - i. Class Structure.
    - ii. Data Structures
    - iii. Algorithms
  - e. Feels a lot like system design.
  - f. Two Approaches (among others): Responsibility Driven Design, Design Patterns
  - g. Note: Discussion of UML here, but see [CS-100-2003-1-NOTES-15]
- II. Responsibility Driven
  - a. Code should be modeled after the problem itself
  - b. Data
    - i. What kind? How much? Number of instances, size of each?
    - ii. What are keys like?
      - 1. If you’re only going to use odd integers, it’s good to know that
      - 2. How will they cluster?
      - 3. In designing a compiler, know that lots of variable names may start with “lpsz” so you’re contemplating using the first four characters of the variable name for something you’ll want to consider that clustering.
  - c. Access Patterns: Will we always need to touch all the objects at the same time? Will most accesses be random?
  - d. Performance
    - i. CPU speed? Data throughput?
    - ii. Can usually trade memory usage for CPU time
  - e. Choosing a Structure
    - i. Know what’s already been done
    - ii. *Art of Programming* (Knuth)
    - iii. Cormen, Lierson, Rivest
    - iv. All programmers should have these
  - f. At what point do you stop designing and start coding?
    - i. It’s easy to start too soon or too late
    - ii. “Make it up as you go” is good for prototyping. Get to play with more options.
    - iii. Err toward more design.
    - iv. If you have writer’s block on the code, you probably haven’t done enough design.
    - v. If you can’t come up with a good design, build a prototype
  - g. Implementing Responsibility Driven Design (Mechanics)
    - i. Seems like something a second or third grader would do, but works really well
    - ii. Take textual description, split into four things:
      - 1. Classes
      - 2. Responsibilities
      - 3. Hierarchy
      - 4. Collaborations – how classes provide implementation for other classes
    - iii. Classes
      - 1. Highlight every noun phrase. Becomes a potential class
      - 2. Adjective phrases *may* turn the noun into a totally different thing
        - a. May be just fluff: “great ____”
        - b. May be an important distinction: “primary” vs. “secondary”
      - 3. Passive voice hides the subject
      - 4. Watch for external subjects (“The User”). May / may not need a class.
    - iv. CRC Cards

1. Create an index card for each class you’ve identified
2. Title is the name of the class
3. Great to tape to big marker board (or at least to a wall)
4. Arrange similar cards together; eliminate duplicates

- v. Rules of Thumb
    1. Model physical objects
    2. Conceptual objects (“the *journey*”)
    3. Categories of classes
    4. Values of attributes (“Color”)
    5. Check for multiple names. Are they the same? Which one (if either) is more appropriate? Getting the name right matters a lot!
- vi. Responsibilities
    1. Highlight all the verb phrases in the text
    2. “remembers” or “stores” indicates responsibility!
    3. The information you need is usually explicit
    4. Look for other stuff too
    5. Assign responsibilities to classes
        - a. Each class has at least one responsibility
        - b. Write responsibilities on the left side of the CRC card
        - c. State as generally as possible. “Finds fastest time” might be just “knows how to search”
        - d. Some responsibilities are associated with more than one class
            - i. Imagine which will be easier to code
            - ii. Keep related behaviors / responsibilities together
        - e. Some don’t seem to belong anywhere.
            - i. Need another class then!
            - ii. Certainly won’t find them all just by highlighting nouns
- vii. Collaborations
    1. Classes aren’t islands
    2. Consider collaborations with classes you’re not writing (like HashTable)
    3. Collaborations go on the right side of the CRC card
- viii. Hierarchy
    1. Superclass names below class names on card
    2. Only use inheritance for “is a” relationships
    3. Don’t use inheritance for implementation sharing
    4. (AddressBook : HashTable is WRONG)
    5. This is where it’s great to have these on the wall
    6. Identify which classes are abstract (may be abstract with no subclass if the subclass will be added later)
    7. Watch out for multiple inheritance

III. Design Patterns

- a. Patterns capture existing expertise
- b. Just a catalog of patterns for how classes interact
- c. See [CS-100-2003-1-NOTES-22]