# 

[Class Notes](#h.qnlokq1xc39f)

[Presentation Notes](#h.fht2vefc47uj)

[Book Notes](#h.rjrhzzl6yp3s)

[Preface & Chapter 1](#h.3lx34wf7k7fa)

[Chapter 2](#h.cbnrssykdes5)

[Assigned Reading](#h.vupv5ni81b4b)

[1/12/16](#h.7pbbxwv2yuqg)

[Don’t Repeat Yourself](#h.pms9b6o0n4cf)

[Orthogonality and the DRY Principle](#h.y9y2gvjuqcnz)

[Building Adaptable Systems](#h.assvuvl2jl0k)

[Tracer Bullets and Prototypes](#h.1xm1puwsfr71)

[Law of Demeter](#h.u9bh3jz1lyb7)

[Using Pattern Languages for Object-Oriented Programs](#h.6fzq3i7d1qef)

# 

# Class Notes

# 

# 

# Presentation Notes

# 

# Book Notes

## Preface & Chapter 1

1. What makes a pragmatic programmer
   1. Early adopter/fast adapter
   2. Inquisitive
   3. Critical thinker
   4. Realistic
   5. Jack of all trades
2. **T-** Care about your craft.**T-** Think about your work
3. Take responsibility
4. **T-** Provide options rather than excuses
5. Entropy / disorder builds up in software
6. **T-** Don’t live with broken windows
   1. Bad designs, decisions, and code
   2. People don’t fight because they think others don’t care.
7. Stone soup analogy
   1. Circumvent start-up fatigue
   2. Tackling a whole system gets complicated
   3. **T-** Be a catalyst for change.
   4. **T-** Remember the big picture
8. Good Enough Software
   1. Involve users in trade-off
   2. **T-** Make quality a requirements issue
   3. Know when to stop
9. Knowledge portfolio
   1. **T-** Invest regularly
   2. Diversify
   3. Manage risk
   4. Buy low, sell high
   5. Review and rebalance
   6. Goals
      1. One new language a year
      2. Read tech and non-tech
      3. Classes
      4. Participate
      5. Experiment with environments
      6. Stay current
      7. Get wired with newsgroups
10. **T-** Critically analyze what you hear
11. Communicate
    1. Understanding an audience
       1. What to teach
       2. Interest level
       3. Sophistication level
       4. Detail desires
       5. Owner of info
       6. Mativation
    2. Choose a style
    3. Make it look good
    4. Get others involved, listen, get back and to people
    5. **T-** It’s both what you say and the way you say it

## Chapter 2

# 

# Assigned Reading

## 1/12/16

### Don’t Repeat Yourself

1. Duplication is waste
2. Repetition in process calls for automation
   1. Manual testing is slow, error-prone, and difficult to repeat
   2. Automatic build processes, test suites, etc
3. Repetition in logic calls for abstraction
4. The principle- Once and only once

### Orthogonality and the DRY Principle

1. Orthogonality - Things that are not conceptually related should not be related in the system
2. Coupling is bad

### Building Adaptable Systems

1. Decisions are not final, abstract
2. Delaying decisions until it’s vital can force abstraction
3. Making code configurable can introduce bugs
   1. But cleaner design and less code
   2. Different kind of bug, can be easier to find and change
4. There’s non-user configurations, like CMS and internal config files
5. Don’t start with framework- “plumb in the least amount of glue you need to make it all work”

### Tracer Bullets and Prototypes

1. Tracer bullet comparison - produce something early to see how close you get
2. Start with skeleton application
3. Feature by feature, use case by use case, you can adjust
4. Build prototypes

### Law of Demeter

1. Limited knowledge other other units
2. Only talk to friends, not strangers
3. More maintainable and adaptable
4. Leads to more wrapper, enlarged interfaces

### Using Pattern Languages for Object-Oriented Programs

1. …

### Point/Counterpoint

1. Design patterns make it too easy to introduce unnecessary complexity into system design and are much too often applied without discipline or experience.
2. Patterns let developers work on OO systems beyond their experience
3. Design patterns are best practices
4. Patterns are crucial to the art and science of software design and programming, rooted in hard-won practice and experience.

## 11/14/16

### Review Practices