BOOK NOTES

OBJECT ORIENTED MODELING AND DESIGN

1. Agile Software Development

1.1. **Agile Practices.**

1.1.1. *The Manifesto of the Agile Alliance:*

- Individuals and interactions $>$ Processes and tools
  - Strong player $\neq$ Ace programmer
  - Tools are necessary, but "better" tools $\neq$ better work. Large, unwieldy tools can be (and often are) just as bad as no tools.
- Working software $>$ Comprehensive documentation
  - Martin's first law of documentation: "Produce no document unless its need is immediate and significant."
- Customer collaboration $>$ Contract negotiation
  - Software $\neq$ commodity. Customer collaboration necessary. Contract must be dynamic because project will be dynamic.
- Responding to change $>$ Following a plan
  - Planning: good. Sticking to an outdated plan: not good. Dynamic plans $\because$ project and requirements are dynamic

1.1.2. *Principles:*

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
  - Early delivery of partially functioning system $\Rightarrow$ high quality of the final system.
  - More frequent delivery $\Rightarrow$ high final quality.
- "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
  - keep going from here

1.1.3. *Practices of Extreme Programming:*

- Customer Team Member
- User Stories
- Short Cycles (software delivery)
- Iteration Plan (plan for software delivery)
- Release Plan (plan for $\approx$ next 6 iterations)
- Acceptance Test (read more about this)
- Pair Programming
- Test-Driven Development

- Collective Ownership
- Continuous Integration
- Sustainable Pace
- Open Workspace
- The Planning Game (devision of responsibility between customers/business people and development people)
- Simple Design
  - Start with the simplest thing that could possibly work ⇒ choose most practical solution that is closest to the ideally simple solution.
  - You aren't going to need it: Fare more often than not, you won't need that thing "one day"... No... Don't waste time on that now.
  - Once and only once: If you find yourself copy-pasting rows of code ⇒ that code needs to be it's own class. Duplicate code is not tolerated.
- Refactoring (code structure degrades/rots and must be frequently refactored)
- Metaphor (consider a solution to a jigsaw puzzle: one solution would be to iterate over the pieced until the matching pieces were found. The metaphor is more powerful. The solution extrapolated from the picture is much better. Big picture over details.)

1.1.4. *Planning:*

- Stopped at p20
- Initial Exploration (identify most significant user stories but don't do *all* stories)
- Spiking, Splitting, and Velocity
  - Large stories tend to be overestimated ⇒ split story into smaller pieces.
  - Small stories tend towards underestimation ⇒ merge them.
  -