**Chapter 34: Themes in Software Craftsmanship**

**Conquer complexity**

* The drive to reduce complexity is at the heart of software development
* Tools
  + Dividing a system into subsystems at the architecture level so that your brain can focus on a smaller amount of the system at one time
  + Carefully defining class interfaces so that you can ignore the internal workings of the class
  + Preserving the abstractions represented by the class interface so that your brain doesn’t have to remember arbitrary details
  + Avoiding global data, because global data vastly increases the percentage of the code that you need to juggle in your brain at any one time
  + Avoiding deep inheritance hierarchies because they are intellectually demanding
  + Avoiding deep nesting loops and conditionals because they can be replaced by simpler control structures that burn up fewer grey cells
  + Carefully defining your approach to error handling rather than using an arbitrary proliferation of different error handling techniques
  + Being systematic about the use of built-in exceptions, which can become a nonlinear control structure that’s difficult to understand
  + Not allowing classes to grow into monster classes that amount to whole programs in themselves
  + Keeping routines short
  + Using clear, self-explanatory variable names so that your brain doesn’t have to waste cycles remembering what I and j stand for
  + Minimizing the number of parameters passed to a routine, or, more importantly, passing only the parameters needed to preserve the routines interface abstraction
  + Using conventions to spare your brain the challenge of remembering arbitrary, accidental differences between different sections of code
  + In general, attacking “accidental difficulties” wherever possible

**Pick Your Process**

* Do your best with requirements
  + Solidify as much as possible
  + Anticipate change
* Refine your methods
* Don’t optimize prematurely

**Write Programs for People first, Computers Second**

* Code readability is key
* Has a positive effect on:
  + Comprehensibility
  + Reviewability
  + Error rate
  + Debugging
  + Modifiability
  + Development time
  + External quality

**Program into Your Language, Not in It**

**Focus Your Attention with the Help of Conventions**

* Conventions are intellectual tools used to manage complexity
* Naming
* Ordering
* Protection against known hazards
* Add predictability
* Can compensate for language weaknesses

**Program in Terms of the Problem Domain**

* Work at the highest level of abstraction
* Separating a Program into Levels of Abstraction
  + Level 0: Operating System Operations and Machine Instructions
    - Usually won’t have to worry about
  + Level 1: Programming Language Structures and Tools
    - Primitive data types, control structures etc
  + Level 2: Low Level Implementation Structures
    - Stacks, queues, linked lists
    - Search algorithms
  + Level 3: Low level Problem Domain Terms
    - Need to understand problem domain vocabulary to assign to code
    - This is the glue layer
  + Level 4: High Level Problem Domain Terms
    - Pretty abstract from the code
    - Should be understandable by nontechs
* Low Level Techniques for Working in the Problem Domain
  + Use classes to implement structures that are meaningful in the problem domain terms
  + Hide information about low level data types and their implementation details
  + Use named constants to document the meaning of strings and of numeric literals
  + Assign intermediate variables to document the results of intermediate calculations
  + Use Boolean functions to clarify complex Boolean test

**Watch for Falling Rocks**

* “Tricky code” is a euphemism for “bad code”
* Make it hard to ignore problems

**Iterate, Repeatedly, Again and Again**

**Religion**

* Don’t get super religious about anything
* Fads come and go
* At one point everyone thought some idea was the end all be all solution to programming languages and software engineering
* Blind faith is bad
* As is adherence to a single method
* Don’t want to become averted to new methodologies and ideas
* Do experiments and don’t get stuck in ruts
* Basically, keep an open mind