**Chapter 25: Code Tuning Strategies**

**Performance Overview**

* Code tuning is one way of improving a programs performance
* You can often find other ways to improve performance more though
  + Ones that take less time
  + And do less harm to the code

Quality Characteristics and Performance

* “More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity”
* Performance is loosely related to code speed

Performance and Code Tuning

* Think about efficiency from each of these viewpoints
  + Program requirements
  + Program design
  + Class and routine design
  + Operating system interactions
  + Code compilation
  + Hardware
  + Code tuning
* Program Requirements
  + Performance is stated as a requirement far more often than it is actually a requirement
  + 1 second to 4 seconds for a user isn’t a big deal, but it costs a ton less for the 4 second version
* Program Design
  + Some program designs make it difficult to write a high performance system, others make it hard not to
  + If you know that a programs size and speed are not important, design the architecture so you can reasonably meet the goals
    - First design a performance-oriented architecture
    - Then a resource-oriented design for individual subsystems, features and classes
* Class and Routine Design
  + Carefully choose
    - Data types
    - Algorithms
  + These affect speed and memory
* Operating System Interactions
  + If working with external files, dynamic memory or output devices, youre dealing with the OS
* Code Compilation
  + Choose the right compiler
  + I guess some are better than others
* Hardware
  + Sometimes you just need better hardware
* Code Tuning
  + Small scale changes that make things run more efficiently

**Introduction to Code Tuning**

* Code tuning is hype but it doesn’t always result in “better” code
* Usually a very very very small percentage of code is responsible for at least half of the slowness
  + 4% -> 50%
  + 20% -> 80%
* Working towards perfection might prevent completion
* Less lines of code != faster code (lmaoo)
* “Premature optimization is the root of all evil”
* Don’t optimize as you go
  + Its “almost impossible” to identify performance bottlenecks before a program is working completely
* **Jackson’s Rules of Optimization:**
  + **Rule 1: Don’t do it**
  + **Rule 2: Don’t do it yet – that is, not until you have a perfectly clear and unoptimized solution**

**Kinds of Fat and Molasses**

Common Sources of Inefficiency

* I/O operations
* Paging
  + Operations that require the OS to swap pages of memory
* System calls
  + IO to disk
  + Keyboard, screen, printer
  + Solutions
    - Write your own services
    - Avoid going to the system
    - Work with the system vendor to make the call faster
* Interpreted language
  + \*cough\* python
* Errors
  + Uncaught errors can have a program doing dumb stuff for a bit of time

**Measurement**

* Measure your code to find hotspots
  + The small percentage responsible for the most slowdowns
* Measurements need to be precise too, or else bad data will lead you astray