**Optimized C++ Summary Notes**

**Chapter 1**

* Book is about how to optimize in development phases either right from the start or in integration/testing and target recording phase. How the code in C++ can be inproved increamentally for fmailiarizing us with code tuning process. Obviously the practice of such measures while writing code will optimize and accustome your approaches.
* In trditional SD environment, Optimization is a coding activity during testing and integration phase when program performance is observed. In Agile, multiple sprints might be allocated for optimization when a feature with performance goal is added in the code.
* Goal-> meet user’s need of speed, throughput, power consumption and memory footprint goals. Poor performace is same kind of problem as bugs and missing features, difference being bugs are discrete in nature (present/absent) but optimization is achieved incrementally.
* Optimization is experimental science. Requires formulation hypothesis based on observations and their testings.
* A huge program is not single handedly coded, therefor there are multiple and significantly high opportunities of optimizations.
* WRONG BELIEFS: Advice on not to optimize has become a widely accepted belief. It takes same amount of time to write efficient code as slow and inefficient code. Why would anyone choose deliberately to write an inefficient piece of work. The received wisdom is not good and its antidode is experimentation.
* Optimize but there, where needed. Only 10% of programs code is performance critical. You need to learn to identify the hot spots in code.
* While time may bring faster computers, it does nothing to improve the performance of the existing hardware. The only speed upgrade a customer will get from the company is a comes from optimizations in subsequent releases. Optimization keeps your program fresh.
* Efficiency matters for an application or a system where it contends for contrained resources.
* Use a better compiler: Use a compiler which produces a much faster executable for the same piece of source code. Use compiler better: One such tip includes use of code optimizer in compiler options. Intels C++ compiler is expensive but produces fastest executables. Intel>Microsoft’s visual C++>GCC.
* Use better Algorithms: This produces the biggest impact on code performance and ersource eficiency. It can give a performance rise of nearly 30% to 100%. Optimizing patterns include precomputation, lazy computation and caching. Such techniques improves the algorithm implementation for a problem, thereby increasing efficiency of the code for a frequently encountered query.
* Use better libraries: Many open source libs that provides sophisticated implementations that are much faster and capable with added features than vendor’s c++ runtime libraries.