

Re: Who is a Good Programmer?

Source: <http://coding.derkeiler.com/Archive/General/comp.programming/2004-04/1732.html>

From: Nick Landsberg (*hukolau_at_NOSPAM.att.net*)

Date: 04/30/04

Date: Fri, 30 Apr 2004 02:19:38 GMT

Robert Wessel wrote:

> *CBFalconer* <cbfalconer@yahoo.com> wrote in message news:<4091300C.F773BC7B@yahoo.com>...

>

>>*Robert Wessel* wrote:

>>

>> ... *snip* ...

>>

>>>*The problem is that many of the goals conflict. Fast code is almost always less reusable, maintainable, and clear, and is usually slower to develop. Reusability tends to slow down development considerably as well. Input validation conflicts with fast and small, and useful error messages assumes some method of handling error messages.*

>>

>>*Splutter. Hogwash. Clean, compact code is usually close to optimum for speed and maintainability, for some definition of clean.*

>

>

>

> *So reading random records from a sequential file via a sequential scan is going to be faster than a (vastly more complex) B-Tree? Quicksort faster than insertion sort? Schoolbook multiplication faster than Karatsuba or FFT? Package Wrapping faster than Gharam scan?*

There is an implication here about the choice of algorithm.

Given that one needed to search (at most) 10 records, the choice of algorithm would probably dictate sequential search. Given 1,000 records, binary search is almost always a better search. Given a million records, go and get a database and don't try to write the code yourself! :)

Once an algorithm has been chosen, then I agree with Chuck that clean (for some arbitrary definition of clean), compact code is easily maintainable.

comp.programming: Re: Who is a Good Programmer?

You are making points at two different levels of abstraction.

- >
- > *Not to mention that most of the low-level optimization techniques tend*
- > *to obfuscate the algorithms pretty badly.*
- >

One of the greatest sins is to apply low-level optimization techniques when high-level ones give more "bang for the buck." A better technique is to look for inefficiencies in your design. As an example, I once saw a program which used 40+ database calls (each of which extracted one field from some table) to populate a screen. The newbie developer was moaning that the database was too slow and how he had to do contortions in his code to get it to run marginally faster. The solution was to use 3-4 database calls which got multiple fields per call. Sounds simple? Yes! But it improved response time by an order of magnitude. Question: Which code would you think is more maintainable? Look to your algorithms! :)

- >
- >
- >> *Unvalidated input is useless, and unhandled errors lead to*
- >> *stochastic processes. Look to your algorithms.*
- >
- >
- >
- > *Input validation can have considerable overhead, in execution time,*
- > *code size, and source code. If you have some guarantees about the*
- > *performance of the user, omitting it may make sense. For example, if*
- > *I know my $\text{sqrt}(x)$ function will only be called from a program that*
- > *generates only positive x 's, it's not unreasonable to omit the range*
- > *checks in a critical path.*

Agreed that it has overhead. Since I work on systems which mostly have other systems as inputs, we have interface agreements with them. On the other hand, we still check the inputs in case they don't live up to the agreements. It is a pain in the neck, but we do validate the inputs because unvalidated inputs will corrupt our data. Core dumping on invalid input is **not** an option.

--

Re: Who is a Good Programmer?

comp.programming: Re: Who is a Good Programmer?

"It is impossible to make anything foolproof
because fools are so ingenious"

- A. Bloch